

CPSVerification

CPSVerification

February 7, 2019

Contents

1	VC_diffKAD	2
1.1	Stack Theories Preliminaries: VC_KAD and ODEs	2
1.2	VC_diffKAD Preliminaries	5
1.2.1	(primed) dSolve preliminaries	5
1.2.2	dInv preliminaries	11
1.2.3	ODE Extras	13
1.3	Phase Space Relational Semantics	16
1.4	Derivation of Differential Dynamic Logic Rules	18
1.4.1	"Differential Weakening"	18
1.4.2	"Differential Cut"	18
1.4.3	"Solve Differential Equation"	19
1.4.4	"Differential Invariant."	26
1.5	Rules Testing	35
2	Flow Locales	40
2.1	Example	45
3	Preliminaries	45
3.1	Weakest Liberal Preconditions	45
3.2	Real Numbers and Derivatives	47
3.3	Vectors, norms and matrices.	50
3.3.1	Unit vectors and vector norm	50
3.3.2	Matrix norm	51
4	Hybrid System Verification	54
4.1	Verification by providing solutions	54
4.2	Verification with differential invariants	55
4.2.1	Differential Weakening	56
4.2.2	Differential Cut	56
4.2.3	Differential Invariant	57
4.3	Examples	65
4.3.1	Specific vector field	65

4.3.2	General vector field	67
4.3.3	Bouncing Ball with solution	70
4.3.4	Bouncing Ball with invariants	71
4.3.5	Circular motion with invariants	72
5	Preliminaries	74
5.1	Nondeterministic Functions	74
5.2	Weakest Liberal Preconditions	76
5.3	Real Numbers and Derivatives	78
5.4	Vectors, norms and matrices.	81
5.4.1	Unit vectors and vector norm	81
5.4.2	Matrix norm	82
6	Hybrid System Verification	85
6.1	Verification by providing solutions	85
6.2	Verification with differential invariants	87
6.2.1	Differential Weakening	87
6.2.2	Differential Cut	88
6.2.3	Differential Invariant	89
6.3	Examples	96
6.3.1	Specific vector field	97
6.3.2	General vector field	99
6.3.3	Bouncing Ball with solution	101
6.3.4	Bouncing Ball with invariants	103
6.3.5	Circular motion with invariants	104

1 VC_diffKAD

```

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

begin

```

1.1 Stack Theories Preliminaries: VC_KAD and ODEs

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

```

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

notation p2r ( $\lceil \cdot \rceil$ )

```

```

and  $r2p$  ( $\lfloor \cdot \rfloor$ )
and  $Set.image$  ( $\lfloor \cdot \rfloor$ )
and  $Product-Type.prod.fst$  ( $\pi_1$ )
and  $Product-Type.prod.snd$  ( $\pi_2$ )
and  $List.zip$  (infixl  $\otimes$  63)
and  $rel\text{-}ad$  ( $\Delta^c_1$ )

```

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  $(x\ solves\text{-}ode\ f)\ \{0..t\}\ R = ((x\ has\text{-}vderiv\text{-}on\ (\lambda t. f\ t\ (x\ t)))\ \{0..t\} \wedge x \in$ 
 $\{0..t\} \rightarrow R)$ 
and  $f \in A \rightarrow B = (f \in \{f. \forall x. x \in A \longrightarrow (f\ x) \in B\})$ 
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  $(x\ has\text{-}vector\text{-}derivative\ x'\ r)\ (at\ r\ within\ \{0..t\}) =$ 
 $(x\ has\text{-}derivative\ (\lambda x. x *_{\mathbb{R}} x'\ r))\ (at\ r\ within\ \{0..t\})$ 
apply( $simp\text{-}all\ add: p2r\text{-}def\ r2p\text{-}def\ rel\text{-}ad\text{-}def\ rel\text{-}antidomain\text{-}kleene\text{-}algebra.fbox\text{-}def$ 

 $solves\text{-}ode\text{-}def\ has\text{-}vderiv\text{-}on\text{-}def$ )
apply( $blast, fastforce, fastforce$ )
using  $has\text{-}vector\text{-}derivative\text{-}def$  by  $auto$ 

```

Observe also, the following consequences and facts:

```

proposition  $\pi_1(\lfloor R \rfloor) = r2s\ R$ 
by ( $simp\ add: fst\text{-}eq\text{-}Domain$ )

```

```

proposition  $\Delta^c_1\ R = Id - \{(s, s) \mid s. s \in (\pi_1(\lfloor R \rfloor))\}$ 
by( $simp\ add: image\text{-}def\ rel\text{-}ad\text{-}def, fastforce$ )

```

```

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

```

```

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

```

```

proposition  $rdom\text{-}p2r\text{-}contents:(a, b) \in rdom\ \lceil P \rceil = ((a = b) \wedge P\ a)$ 
proof–
have  $(a, b) \in rdom\ \lceil P \rceil = ((a = b) \wedge (a, a) \in rdom\ \lceil P \rceil)$  using  $p2r\text{-}subid$  by
 $fastforce$ 
also have  $\dots = ((a = b) \wedge (a, a) \in \lceil P \rceil)$  by  $simp$ 
also have  $\dots = ((a = b) \wedge P\ a)$  by ( $simp\ add: p2r\text{-}def$ )

```

ultimately show *?thesis* by simp
qed

~~//Should not add these complement rule's to simp//~~

proposition *rel-ad-rule1*: $(x, x) \notin \Delta^c_1 [P] \implies P\ x$
by(auto simp: *rel-ad-def* *p2r-subid* *p2r-def*)

proposition *rel-ad-rule2*: $(x, x) \in \Delta^c_1 [P] \implies \neg P\ x$
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*: $R \subseteq Id \implies (x, x) \notin R \implies (x, x) \in \Delta^c_1 R$
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*: $(x, x) \in R \implies (x, x) \notin \Delta^c_1 R$
by(metis *empty-iff* *rel-antidomain-kleene-algebra.addual.ars1* *relcomp.relcompI*)

proposition *boxProgrPred-chrcrtrzn*: $(x, x) \in wp\ R\ [P] = (\forall\ y. (x, y) \in R \longrightarrow P\ y)$
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 $d\ p \leq |x^*|\ q$
proof–
from $\langle d\ i \leq |x|\ i \rangle$ **have** $d\ i \leq |x|\ (d\ i)$
using *local.fbox-simp* **by** auto
hence $|1|\ p \leq |x^*|\ i$ **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*:
 $(x, y) \# tail = xList \otimes yList \implies \exists xTail\ yTail. x \# xTail = xList \wedge y \# yTail = yList$
by(*induction* *xList*, *simp-all*, *induction* *yList*, *simp-all*)

proposition *set-zip-left-rightD*:
 $(x, y) \in set\ (xList \otimes yList) \implies x \in set\ xList \wedge y \in set\ yList$
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*]

1.2 VC_diffKAD Preliminaries

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

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

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

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

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

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

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

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

1.2.1 (primed) dSolve preliminaries

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

abbreviation $varDiffs-to-zero :: real store \Rightarrow real store$ (sol) **where**
 $sol\ a \equiv (override-on\ a\ (\lambda x. 0)\ varDiffs)$

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

proposition $varDiffs-to-zero-beginning[simp]: take\ 2\ x \neq "d[" \implies (sol\ s)\ x = s$
 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 $vderiv\text{-}of\ f\ S = (SOME\ f'.\ (f\ \text{has-}vderiv\text{-}on\ f')\ S)$

primrec $state\text{-}list\text{-}upd :: ((real \Rightarrow real\ store \Rightarrow real) \times string \times (real\ store \Rightarrow real))\ list \Rightarrow real \Rightarrow real\ store \Rightarrow real\ store$ **where**
 $state\text{-}list\text{-}upd\ []\ t\ s = s$
 $state\text{-}list\text{-}upd\ (uxf\ \#\ tail)\ t\ s = (state\text{-}list\text{-}upd\ tail\ t\ s)$
 $(\pi_1\ (\pi_2\ uxf)) := (\pi_1\ uxf)\ t\ s,$
 $\partial\ (\pi_1\ (\pi_2\ uxf)) := (if\ t = 0\ then\ (\pi_2\ (\pi_2\ uxf))\ s$
 $else\ vderiv\text{-}of\ (\lambda\ r.\ (\pi_1\ uxf)\ r\ s)\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t))$

abbreviation $state\text{-}list\text{-}cross\text{-}upd :: real\ store \Rightarrow (string \times (real\ store \Rightarrow real))\ list \Rightarrow (real \Rightarrow real\ store \Rightarrow real)\ list \Rightarrow real \Rightarrow (char\ list \Rightarrow real)\ (-[\leftarrow] - [64,64,64] 63)$ **where**
 $s[xfList \leftarrow uInput]\ t \equiv state\text{-}list\text{-}upd\ (uInput \otimes xfList)\ t\ s$

proposition $state\text{-}list\text{-}cross\text{-}upd\text{-}empty[simp]: (s[\leftarrow list]\ t) = s$
by(*induction list, simp-all*)

lemma *inductive-state-list-cross-upd-its-vars:*

assumes $distHyp: distinct\ (map\ \pi_1\ ((y, g) \# xftail))$
and $varHyp: \forall xf \in set((y, g) \# xftail). \pi_1\ xf \notin varDiffs$
and $indHyp: (u, x, f) \in set\ (utail \otimes xftail) \implies (s[xftail \leftarrow utail]\ t)\ x = u\ t\ s$
and $disjHyp: (u, x, f) = (v, y, g) \vee (u, x, f) \in set\ (utail \otimes xftail)$
shows $(s[(y, g) \# xftail \leftarrow v \# utail]\ t)\ x = u\ t\ s$
using $disjHyp$ **proof**
 $assume\ (u, x, f) = (v, y, g)$
 $hence\ (s[(y, g) \# xftail \leftarrow v \# utail]\ t)\ x = ((s[xftail \leftarrow utail]\ t)(x := u\ t\ s,$
 $\partial\ x := if\ t = 0\ then\ f\ s\ else\ vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t))\ x$ **by**
 $simp$
 $also\ have\ \dots = u\ t\ s$ **by** (*simp add: vdiff-def*)
 $ultimately\ show\ ?thesis$ **by** $simp$
next
 $assume\ yTailHyp: (u, x, f) \in set\ (utail \otimes xftail)$
from $this$ **and** $indHyp$ **have** $3: (s[xftail \leftarrow utail]\ t)\ x = u\ t\ s$ **by** $fastforce$
from $yTailHyp$ **and** $distHyp$ **have** $2: y \neq x$ **using** $set\text{-}zip\text{-}left\text{-}rightD$ **by** $force$
from $yTailHyp$ **and** $varHyp$ **have** $1: x \neq \partial\ y$
using $set\text{-}zip\text{-}left\text{-}rightD\ vdiff\text{-}invarDiffs$ **by** $fastforce$
from 1 **and** 2 **have** $(s[(y, g) \# xftail \leftarrow v \# utail]\ t)\ x = (s[xftail \leftarrow utail]\ t)\ x$
by $simp$
 $thus\ ?thesis$ **using** 3 **by** $simp$
qed

theorem *state-list-cross-upd-its-vars:*

assumes $distinctHyp: distinct\ (map\ \pi_1\ xfList)$
and $lengthHyp: length\ xfList = length\ uInput$
and $varsHyp: \forall xf \in set\ xfList. \pi_1\ xf \notin varDiffs$
and $its\text{-}var: (u, x, f) \in set\ (uInput \otimes xfList)$

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*: $x \in X \implies (\text{override-on } f \ g \ X)(x := z) = (\text{override-on } f \ (g(x := z)) \ X)$
by (*rule ext*, *simp add: override-on-def*)

lemma *inductive-state-list-cross-upd-its-dvars*:
assumes $\exists g. (s[xfTail \leftarrow uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
and $\forall xf \in \text{set} \ (xf \ \# \ xfTail). \ \pi_1 \ xf \notin \text{varDiffs}$
and $\forall uxf \in \text{set} \ (u \ \# \ uTail \otimes xf \ \# \ xfTail). \ \pi_1 \ uxf \ 0 \ s = s \ (\pi_1 \ (\pi_2 \ uxf))$
shows $\exists g. (s[xf \ \# \ xfTail \leftarrow u \ \# \ uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
proof –
let $?gLHS = (s[(xf \ \# \ xfTail) \leftarrow (u \ \# \ uTail)] \ 0)$
have *observ*: $\partial (\pi_1 \ xf) \in \text{varDiffs}$ **by** (*auto simp: varDiffs-def*)
from *assms*(1) **obtain** *g* **where** $(s[xfTail \leftarrow uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
by *force*
then have $?gLHS = (\text{override-on } s \ g \ \text{varDiffs})(\pi_1 \ xf := u \ 0 \ s, \ \partial (\pi_1 \ xf) := \pi_2 \ xf \ s)$ **by** *simp*
also have $\dots = (\text{override-on } s \ g \ \text{varDiffs})(\partial (\pi_1 \ xf) := \pi_2 \ xf \ s)$
using *override-on-def* *varDiffs-def* *assms* **by** *auto*
also have $\dots = (\text{override-on } s \ (g(\partial (\pi_1 \ xf) := \pi_2 \ xf \ s)) \ \text{varDiffs})$
using *observ* **and** *override-on-upd* **by** *force*
ultimately show *?thesis* **by** *auto*
qed

theorem *state-list-cross-upd-its-dvars*:
assumes *lengthHyp*: $\text{length } xfList = \text{length } uInput$
and *varsHyp*: $\forall \ xf \in \text{set } xfList. \ \pi_1 \ xf \notin \text{varDiffs}$
and *solHyp1*: $\forall \ uxf \in \text{set} \ (uInput \otimes xfList). \ (\pi_1 \ uxf) \ 0 \ s = s \ (\pi_1 \ (\pi_2 \ uxf))$
shows $\exists \ g. (s[xfList \leftarrow uInput] \ 0) = (\text{override-on } s \ g \ \text{varDiffs})$
using *assms* **proof**(*induct* *xfList* *uInput* *rule: list-induct2'*)
case 1
have $(s[\] \leftarrow \] \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *metis*
next
case (2 *xf xfTail*)
have $(s[(xf \ \# \ xfTail) \leftarrow \] \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *metis*
next
case (3 *u utail*)
have $(s[\] \leftarrow \ utail \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *force*
next

case (λ xf $xfTail$ u $uTail$)
 then have $\exists g. (s[xfTail \leftarrow uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$ by simp
 thus ?case using inductive-state-list-cross-upd-its-dvars λ .prems by blast
 qed

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

lemma *closed-vderiv-on-cong-to-open-vderiv*:
 assumes $gHyp : \forall \tau > 0. f \ \tau = g \ \tau$
 and $fHyp : \forall t \geq 0. (f \text{ has-vderiv-on } f') \ \{0 \cdot t\}$
 and $tHyp : t > 0$ and $cHyp : c > 1$
 shows $\text{vderiv-of } g \ \{0 < \cdot < (c *_{\mathbb{R}} t)\} \ t = f' \ t$
proof–
 have $ctHyp : c \cdot t > 0$ using $tHyp$ and $cHyp$ by auto
 from $fHyp$ have $(f \text{ has-vderiv-on } f') \ \{0 < \cdot < c \cdot t\}$ using has-vderiv-on-subset
 by (metis greaterThanLessThan-subseteq-atLeastAtMost-iff less-eq-real-def)
 then have $\text{derivHyp} : (g \text{ has-vderiv-on } f') \ \{0 < \cdot < c \cdot t\}$
 using $gHyp$ $ctHyp$ and has-vderiv-on-cong-open-interval by blast
 hence $f'Hyp : \forall f''. (g \text{ has-vderiv-on } f'') \ \{0 < \cdot < c \cdot t\} \longrightarrow (\forall \tau \in \{0 < \cdot < c \cdot t\}. f' \ \tau = f'' \ \tau)$
 using vderiv-unique-within-open-interval $ctHyp$ by blast
 also have $(g \text{ has-vderiv-on } (\text{vderiv-of } g \ \{0 < \cdot < (c *_{\mathbb{R}} t)\})) \ \{0 < \cdot < c \cdot t\}$
 by (simp add: vderiv-of-def, metis derivHyp someI-ex)
 ultimately show $\text{vderiv-of } g \ \{0 < \cdot < c *_{\mathbb{R}} t\} \ t = f' \ t$ using $tHyp$ $cHyp$ by force
 qed

lemma *vderiv-of-to-sol-its-vars*:
 assumes $\text{distinctHyp} : \text{distinct } (\text{map } \pi_1 \ \text{xfList})$
 and $\text{lengthHyp} : \text{length } \text{xfList} = \text{length } uInput$
 and $\text{varsHyp} : \forall xf \in \text{set } \text{xfList}. \pi_1 \ xf \notin \text{varDiffs}$
 and $\text{solHyp2} : \forall t \geq 0. ((\lambda \tau. (\text{sol } s[\text{xfList} \leftarrow uInput] \ \tau) \ x) \text{ has-vderiv-on } (\lambda \tau. f \ (\text{sol } s[\text{xfList} \leftarrow uInput] \ \tau))) \ \{0 \cdot t\}$

and $tHyp: t > 0$ **and** $uxfHyp: (u, x, f) \in \text{set } (uInput \otimes xfList)$
shows $vderiv\text{-}of (\lambda\tau. u \tau (sol\ s)) \{0 < .. < (2 *_{\mathcal{R}} t)\} \ t = f (sol\ s[xfList \leftarrow uInput]$
 $t)$
apply($rule\text{-}tac\ f = (\lambda\tau. (sol\ s[xfList \leftarrow uInput] \tau) \ x)$ **in** $closed\text{-}vderiv\text{-}on\text{-}cong\text{-}to\text{-}open\text{-}vderiv)$
subgoal using $assms$ **and** $state\text{-}list\text{-}cross\text{-}upd\text{-}its\text{-}vars$ **by** $metis$
by($simp\text{-}all\ add: solHyp2\ tHyp$)

lemma $inductive\text{-}to\text{-}sol\text{-}zero\text{-}its\text{-}dvars$:

assumes $eqFuncs: \forall\ s. \forall\ g. \forall\ xf \in \text{set } ((x, f) \# xfs). \pi_2\ xf\ (override\text{-}on\ s\ g\ varDiffs)$
 $= \pi_2\ xf\ s$
and $eqLengths: length\ ((x, f) \# xfs) = length\ (u \# us)$
and $distinct: distinct\ (map\ \pi_1\ ((x, f) \# xfs))$
and $vars: \forall\ xf \in \text{set } ((x, f) \# xfs). \pi_1\ xf \notin varDiffs$
and $solHyp1: \forall\ uxf \in \text{set } ((u \# us) \otimes ((x, f) \# xfs)). \pi_1\ uxf\ 0\ (sol\ s) = sol\ s\ (\pi_1\ (\pi_2\ uxf))$
and $disjHyp: (y, g) = (x, f) \vee (y, g) \in \text{set } xfs$
and $indHyp: (y, g) \in \text{set } xfs \implies (sol\ s[xfs \leftarrow us]\ 0) (\partial\ y) = g\ (sol\ s[xfs \leftarrow us]\ 0)$
shows $(sol\ s[(x, f) \# xfs \leftarrow u \# us]\ 0) (\partial\ y) = g\ (sol\ s[(x, f) \# xfs \leftarrow u \# us]\ 0)$
proof –
from $assms$ **obtain** $h1$ **where** $h1Def: (sol\ s[((x, f) \# xfs) \leftarrow (u \# us)]\ 0) =$
 $(override\text{-}on\ (sol\ s)\ h1\ varDiffs)$ **using** $state\text{-}list\text{-}cross\text{-}upd\text{-}its\text{-}dvars$ **by** $blast$
from $disjHyp$ **show** $(sol\ s[(x, f) \# xfs \leftarrow u \# us]\ 0) (\partial\ y) = g\ (sol\ s[(x, f) \#$
 $xfs \leftarrow u \# us]\ 0)$
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 $... = (sol\ s[(x, f) \# xfs \leftarrow u \# us]\ 0) (\partial\ y)$ **using** $eqHeads$ **by** $auto$
ultimately show $?thesis$ **by** $linarith$

next

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

lemma $to\text{-}sol\text{-}zero\text{-}its\text{-}dvars$:

assumes $funcsHyp: \forall\ s. \forall\ g. \forall\ xf \in \text{set } xfList. \pi_2\ xf\ (override\text{-}on\ s\ g\ varDiffs)$
 $= \pi_2\ xf\ s$

and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 xf \notin \text{varDiffs}$
and *solHyp1*: $\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1 uxf) \ 0 \ (sol \ s) = (sol \ s) \ (\pi_1 \ (\pi_2 \ uxf))$
and *ygHyp*: $(y, g) \in \text{set } xfList$
shows $(sol \ s[xfList \leftarrow uInput] \ 0)(\partial \ y) = g \ (sol \ s[xfList \leftarrow uInput] \ 0)$
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* $((y, g) \# xfs) = \text{length } (v \# vs)$
and *distHyp*:*distinct* (*map* π_1 $((y, g) \# xfs)$)
and *varHyp*: $\forall xf \in \text{set } ((y, g) \# xfs). \pi_1 xf \notin \text{varDiffs}$
and *indHyp*: $(u, x, f) \in \text{set } (vs \otimes xfs) \implies (s[xfs \leftarrow vs]t)(\partial \ x) = vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t$
and *disjHyp*: $(v, y, g) = (u, x, f) \vee (u, x, f) \in \text{set } (vs \otimes xfs)$ **and** *tHyp*: $t > 0$
shows $(s[(y, g) \# xfs \leftarrow v \# vs] \ t) \ (\partial \ x) = vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t$
proof–
let *?lhs* = $((s[xfs \leftarrow vs] \ t)(y := v \ t \ s, \partial \ y := vderiv-of \ (\lambda r. v \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t)) \ (\partial \ x)$
let *?rhs* = $vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t$
have $(s[(y, g) \# xfs \leftarrow v \# vs] \ t) \ (\partial \ x) = ?lhs$ **using** *tHyp* **by** *simp*
also have $vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t = ?rhs$ **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* = $vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t$ **by** *simp*
also have $vderiv-of \ (\lambda r. u \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t = ?rhs$ **using** *uxfEq* **by** *simp*
ultimately show *?lhs* = *?rhs* **by** *simp*
next
assume *sygTail*: $(u, x, f) \in \text{set } (vs \otimes xfs)$
from *this* **have** $y \neq x$ **using** *distHyp* *set-zip-left-rightD* **by** *force*
hence $\partial \ x \neq \partial \ y$ **by**(*simp* *add*: *vdiff-def*)
have $y \neq \partial \ x$ **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
then have *?lhs* = $(s[xfs \leftarrow vs] \ t) \ (\partial \ x)$ **using** $\langle y \neq \partial \ x \rangle$ **and** $\langle \partial \ x \neq \partial \ y \rangle$ **by** *simp*
also have $(s[xfs \leftarrow vs] \ t) \ (\partial \ x) = ?rhs$ **using** *indHyp* *sygTail* **by** *simp*
ultimately show *?lhs* = *?rhs* **by** *simp*
qed
from *this* **and** *obs* **show** *?thesis* **by** *simp*
qed

lemma *to-sol-greater-than-zero-its-dvars*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 xf \notin \text{varDiffs}$
and *uxfHyp*: $(u, x, f) \in \text{set } (uInput \otimes xfList)$ **and** *tHyp*: $t > 0$

shows ($s[xfList \leftarrow uInput]$ t) (∂x) = $vderiv\text{-}of$ ($\lambda r. u\ r\ s$) $\{0 < .. < (2 *_{\mathbb{R}} t)\}$ 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*)

1.2.2 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* ($\partial_t - [54]$ 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* ($\partial_P - [54]$ 70) **where**

$(\partial_P\ (\vartheta \doteq \eta)) = ((\partial_t\ \vartheta) \doteq (\partial_t\ \eta))$
 $(\partial_P\ (\vartheta \prec \eta)) = ((\partial_t\ \vartheta) \prec (\partial_t\ \eta))$
 $(\partial_P\ (\vartheta \preceq \eta)) = ((\partial_t\ \vartheta) \preceq (\partial_t\ \eta))$
 $(\partial_P\ (\varphi \sqcap \psi)) = (\partial_P\ \varphi) \sqcap (\partial_P\ \psi)$
 $(\partial_P\ (\varphi \sqcup \psi)) = (\partial_P\ \varphi) \sqcup (\partial_P\ \psi)$

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**

$xtList\ \langle t_C\ r \rangle = t_C\ r$
 $\llbracket \langle t_V\ x \rangle = t_V\ x \rrbracket$
 $((y, \xi) \# xtTail)\langle Var\ x \rangle = (if\ x = y\ then\ \xi\ else\ xtTail\langle Var\ x \rangle)$
 $xtList\ \langle \ominus\ \vartheta \rangle = \ominus\ (xtList\ \langle \vartheta \rangle)$
 $xtList\ \langle \vartheta \oplus \eta \rangle = (xtList\ \langle \vartheta \rangle) \oplus (xtList\ \langle \eta \rangle)$
 $xtList\ \langle \vartheta \odot \eta \rangle = (xtList\ \langle \vartheta \rangle) \odot (xtList\ \langle \eta \rangle)$

proposition $substList\text{-on-compl-of-varDiffs}$:

assumes $trmVars\ \eta \subseteq (UNIV - varDiffs)$

and $set\ (map\ \pi_1\ xtList) \subseteq varDiffs$

shows $xtList\ \langle \eta \rangle = \eta$

using $assms\ apply\ (induction\ \eta,\ simp\text{-all}\ add:\ varDiffs\text{-def})$

by $(induction\ xtList,\ auto)$

lemma $substList\text{-help1} :: set\ (map\ \pi_1\ ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes\ uInput)) \subseteq varDiffs$

apply $(induct\ xfList\ uInput\ rule:\ list\text{-induct2}',\ simp\text{-all}\ add:\ varDiffs\text{-def})$

by $auto$

lemma $substList\text{-help2}$:

assumes $trmVars\ \eta \subseteq (UNIV - varDiffs)$

shows $((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes\ uInput)\langle \eta \rangle = \eta$

using $assms\ substList\text{-help1}\ substList\text{-on-compl-of-varDiffs}\ \mathbf{by}\ blast$

lemma $substList\text{-cross-vdiff-on-non-occurring-var}$:

assumes $x \notin set\ list1$

shows $((map\ vdiff\ list1) \otimes\ list2)\langle t_V\ (\partial\ x) \rangle = t_V\ (\partial\ x)$

using $assms\ apply\ (induct\ list1\ list2\ rule:\ list\text{-induct2}',\ simp,\ simp,\ clarsimp)$

by $(simp\ add:\ vdiff\text{-def})$

primrec $propVars :: props \Rightarrow string\ set$ **where**

$propVars\ (\vartheta \doteq \eta) = trmVars\ \vartheta \cup trmVars\ \eta$
 $propVars\ (\vartheta \prec \eta) = trmVars\ \vartheta \cup trmVars\ \eta$
 $propVars\ (\vartheta \preceq \eta) = trmVars\ \vartheta \cup trmVars\ \eta$
 $propVars\ (\varphi \sqcap \psi) = propVars\ \varphi \cup propVars\ \psi$
 $propVars\ (\varphi \sqcup \psi) = propVars\ \varphi \cup propVars\ \psi$

primrec $subspList :: (string \times trms)\ list \Rightarrow props \Rightarrow props\ (-\vdash\ [54]\ 80)$ **where**

$xtList\ \vdash\ \vartheta \doteq \eta \vdash = ((xtList\ \langle \vartheta \rangle) \doteq (xtList\ \langle \eta \rangle))$
 $xtList\ \vdash\ \vartheta \prec \eta \vdash = ((xtList\ \langle \vartheta \rangle) \prec (xtList\ \langle \eta \rangle))$

$$\begin{aligned}
xtList \vdash \vartheta \preceq \eta &= ((xtList \langle \vartheta \rangle) \preceq (xtList \langle \eta \rangle)) \\
xtList \vdash \varphi \sqcap \psi &= ((xtList \vdash \varphi) \sqcap (xtList \vdash \psi)) \\
xtList \vdash \varphi \sqcup \psi &= ((xtList \vdash \varphi) \sqcup (xtList \vdash \psi))
\end{aligned}$$

1.2.3 ODE Extras

For exemplification purposes, we compile some concrete derivatives used commonly in classical mechanics. A more general approach should be taken that generates these 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*: $((\cdot) \text{ a has-vector-derivative } a) \text{ (at } x \text{ within } T)$

by (*auto intro: derivative-eq-intros*)

lemma [*poly-deriv*]: $((\cdot) \text{ a has-derivative } (\lambda x. x *_R a)) \text{ (at } x \text{ within } T)$

using *vector-derivative-line-at-origin* **unfolding** *has-vector-derivative-def* **by** *simp*

lemma *quadratic-monomial-derivative*:

$((\lambda t::\text{real}. a \cdot t^2) \text{ has-derivative } (\lambda t. a \cdot (2 \cdot x \cdot t))) \text{ (at } x \text{ within } T)$

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*:

$((\lambda t::\text{real}. a \cdot t^2 / 2) \text{ has-derivative } (\lambda t. a \cdot x \cdot t)) \text{ (at } x \text{ within } T)$

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(190)*)
apply(*rule-tac f'1= $\lambda x. a \cdot x$ and g'1= $\lambda x. v$ in derivative-intros(190)*)
using *poly-deriv(2)* **by**(*auto intro: derivative-intros*)

lemma [*poly-deriv*]:
 $t \in T \implies ((\lambda \tau. a \cdot \tau^2 / 2 + v \cdot \tau + x)$ *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*]:
 $t > 0 \implies vderiv-of\ (\lambda t. a \cdot t^2 / 2 + v \cdot t + x)\ \{0 <..< 2 \cdot t\}\ t = a \cdot t + v$
proof–
let $?f = vderiv-of\ (\lambda t. a \cdot t^2 / 2 + v \cdot t + x)\ \{0 <..< 2 \cdot t\}$
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 $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x)$ *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 $(vderiv-of\ (\lambda t. a \cdot t^2 / 2 + v \cdot t + x)\ \{0 <..< 2 \cdot t\})\ t = a \cdot t + v$
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 $t > 0 \implies vderiv-of\ (\lambda t. a \cdot t^2 / 2 + v \cdot t + x)\ \{0 <..< 2 \cdot t\}\ t = a \cdot t + v$
unfolding *vderiv-of-def* **apply**(*subst someI-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(190)*)

unfolding *has-vderiv-on-def* **by**(*auto intro: derivative-eq-intros*)

lemma [*galilean-transform-eq*]:

$t > 0 \implies \text{vderiv-of } (\lambda r. a \cdot r + v) \{0 < .. < 2 \cdot t\} t = a$

proof –

let $?f = \text{vderiv-of } (\lambda r. a \cdot r + v) \{0 < .. < 2 \cdot t\}$

assume $t > 0$ **hence** $t \in \{0 < .. < 2 \cdot t\}$ **by** *auto*

have $\exists f. ((\lambda r. a \cdot r + v) \text{ has-vderiv-on } f) \{0 < .. < 2 \cdot t\}$

using *galilean-velocity* **by** *blast*

hence $((\lambda r. a \cdot r + v) \text{ has-vderiv-on } ?f) \{0 < .. < 2 \cdot t\}$

unfolding *vderiv-of-def* **by** (*metis (mono-tags, lifting) someI-ex*)

also have $((\lambda r. a \cdot r + v) \text{ has-vderiv-on } (\lambda t. a)) \{0 < .. < 2 \cdot t\}$

using *galilean-velocity* **by** *simp*

ultimately show $(\text{vderiv-of } (\lambda r. a \cdot r + v) \{0 < .. < 2 \cdot t\}) t = a$

apply(*rule-tac f'=?f and $\tau=t$ and $t=2 \cdot t$ in vderiv-unique-within-open-interval*)

using $\langle t \in \{0 < .. < 2 \cdot t\} \rangle$ **by** *auto*

qed

lemma [*galilean-transform*]:

$((\lambda t. v \cdot t - a \cdot t^2 / 2 + x) \text{ has-vderiv-on } (\lambda x. v - a \cdot x)) \{0..t\}$

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*]: $t > 0 \implies \text{vderiv-of } (\lambda t. v \cdot t - a \cdot t^2 / 2 + x) \{0 < .. < 2 \cdot t\} t = v - a \cdot t$

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*]:

$((\lambda t. v - a \cdot t) \text{ has-vderiv-on } (\lambda x. - a)) \{0..t\}$

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*]: $t > 0 \implies \text{vderiv-of } (\lambda r. v - a \cdot r) \{0 < .. < 2 \cdot t\} t = - a$

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*]: $(\lambda x. \text{case } x \text{ of } (t, x) \Rightarrow f t) = (\lambda x. (f \circ \pi_1) x)$

by *auto*

end

theory *VC-diffKAD*

imports *VC-diffKAD-auxiliarities*

begin

1.3 Phase Space Relational Semantics

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

lemma *solves-store-ivpI*:

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

lemma [*solves-store-ivpE*]:

assumes $\varphi_S \text{ solvesTheStoreIVP } xfList \text{ withInitState } s$
shows $\forall y. y \notin \text{varDiffs} \longrightarrow \varphi_S 0 y = s y$
proof(*clarify, rename-tac x*)
fix *x* **assume** $x \notin \text{varDiffs}$
from *assms* **and** *solves-store-ivpE*(5) **have** $x \in (\pi_1(\text{set } xfList)) \Longrightarrow \varphi_S 0 x = s$
x **by** *fastforce*
also have $x \notin (\pi_1(\text{set } xfList)) \cup \text{varDiffs} \Longrightarrow \varphi_S 0 x = s x$
using *assms* **and** *solves-store-ivpE*(1) **by** *simp*

ultimately show $\varphi_S \ 0 \ x = s \ x$ using $\langle x \notin \text{varDiffs} \rangle$ 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 $y \notin (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs}$
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 $y \notin (\pi_1(\text{set } \text{xfList}))$
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 $xf \in \text{set } \text{xfList}$
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 $xf \in \text{set } \text{xfList}$
shows $((\lambda \ t. \ \varphi_S \ t \ (\pi_1 \ xf)) \text{ solves-ode } (\lambda \ t. \lambda \ r. (\pi_2 \ xf) \ (\varphi_S \ t))) \ \{0..t\} \ \text{UNIV}$
using *assms solves-store-ivpE(4)* by *simp*

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

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

$\wedge \text{ solvesStoreIVP } \varphi_S \text{ xflist } s\}$

1.4 Derivation of Differential Dynamic Logic Rules

1.4.1 "Differential Weakening"

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

theorem *dWeakening*:

assumes *guardImpliesPost*: $\lceil G \rceil \subseteq \lceil Q \rceil$

shows $\text{PRE } P \text{ (ODEsystem } \text{xflist with } G) \text{ POST } Q$

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*: $\text{wp } (\text{ODEsystem } \text{xflist with } G) \lceil Q \rceil = \text{wp } (\text{ODEsystem } \text{xflist with } G) \lceil \lambda s. G \ s \longrightarrow Q \ s \rceil$

unfolding *rel-antidomain-kleene-algebra.fbox-def rel-ad-def guarDiffEqtn-def*
by (*simp add: relcomp.simps p2r-def, fastforce*)

1.4.2 "Differential Cut"

lemma *all-interval-guarDiffEqtn*:

assumes *solvesStoreIVP* $\varphi_S \text{ xflist } s \wedge (\forall r \in \{0..t\}. G (\varphi_S \ r)) \wedge 0 \leq t$

shows $\forall r \in \{0..t\}. (s, \varphi_S \ r) \in (\text{ODEsystem } \text{xflist with } G)$

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*: $(s, s) \in \text{wp } (\text{ODEsystem } \text{xflist with } G) \lceil C \rceil$

and *FisSol:solvesStoreIVP* $\varphi_S \text{ xflist } s \wedge (\forall r \in \{0..t\}. G (\varphi_S \ r)) \wedge 0 \leq t$

shows $\forall r \in \{0..t\}. G (\varphi_S \ r) \wedge C (\varphi_S \ r)$

proof—

from *boxDiffC* **have** $\forall c. (s, c) \in (\text{ODEsystem } \text{xflist with } G) \longrightarrow C \ c$

by (*simp add: boxProgrPred-chrctrzn*)

also from *FisSol* **have** $\forall r \in \{0..t\}. (s, \varphi_S \ r) \in (\text{ODEsystem } \text{xflist with } G)$

using *all-interval-guarDiffEqtn* **by** *blast*

ultimately show *?thesis*

using *FisSol atLeastAtMost-iff guarDiffEqtn-def* **by** *fastforce*

qed

theorem *dCut*:

assumes *pBoxDiffCut*: $(\text{PRE } P \text{ (ODEsystem } \text{xflist with } G) \text{ POST } C)$

assumes *pBoxCutQ*: $(\text{PRE } P \text{ (ODEsystem } \text{xflist with } (\lambda s. G \ s \wedge C \ s)) \text{ POST } Q)$

shows $\text{PRE } P \text{ (ODEsystem } \text{xflist with } G) \text{ POST } Q$

apply (*clarify, subgoal-tac a = b*) **defer**

proof (*metis d-p2r rdom-p2r-contents, simp, subst boxProgrPred-chrctrzn, clarify*)

fix *b y* **assume** $(b, b) \in \lceil P \rceil$ **and** $(b, y) \in \text{ODEsystem } \text{xflist with } G$

then obtain $\varphi_S t$ **where** $*:solvesStoreIVP \ \varphi_S \ xfList \ b \wedge (\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r)) \wedge 0 \leq t \wedge \varphi_S \ t = y$
using *guarDiffEqtn-def* **by** *auto*
hence $\forall \ r \in \{0..t\}. (b, \varphi_S \ r) \in (ODEsystem \ xfList \ \text{with} \ G)$
using *all-interval-guarDiffEqtn* **by** *blast*
from this and *pBoxDiffCut* **have** $\forall \ r \in \{0..t\}. C \ (\varphi_S \ r)$
using *boxProgrPred-chrcrtrzn* $((b, b) \in [P])$ **by** *(metis (no-types, lifting) d-p2r subsetCE)*
then have $\forall \ r \in \{0..t\}. (b, \varphi_S \ r) \in (ODEsystem \ xfList \ \text{with} \ (\lambda \ s. \ G \ s \wedge C \ s))$
using ** all-interval-guarDiffEqtn* **by** *(metis (mono-tags, lifting))*
from this and *pBoxCutQ* **have** $\forall \ r \in \{0..t\}. Q \ (\varphi_S \ r)$
using *boxProgrPred-chrcrtrzn* $((b, b) \in [P])$ **by** *(metis (no-types, lifting) d-p2r subsetCE)*
thus $Q \ y$ **using** *** **by** *auto*
qed

theorem *dC*:

assumes $Id \subseteq wp \ (ODEsystem \ xfList \ \text{with} \ G) \ [C]$
shows $wp \ (ODEsystem \ xfList \ \text{with} \ G) \ [Q] = wp \ (ODEsystem \ xfList \ \text{with} \ (\lambda \ s. \ G \ s \wedge C \ s)) \ [Q]$
proof(*rule-tac f= $\lambda \ x. \ wp \ x \ [Q]$ in HOL.arg-cong, safe*)
fix $a \ b$ **assume** $(a, b) \in ODEsystem \ xfList \ \text{with} \ G$
then obtain $\varphi_S t$ **where** $*:solvesStoreIVP \ \varphi_S \ xfList \ a \wedge (\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r)) \wedge 0 \leq t \wedge \varphi_S \ t = b$
using *guarDiffEqtn-def* **by** *auto*
hence $1:\forall \ r \in \{0..t\}. (a, \varphi_S \ r) \in ODEsystem \ xfList \ \text{with} \ G$
by *(meson all-interval-guarDiffEqtn)*
from this have $\forall \ r \in \{0..t\}. C \ (\varphi_S \ r)$ **using** *assms boxProgrPred-chrcrtrzn*
by *(metis IdI boxProgrPred-IsProp subset-antisym)*
thus $(a, b) \in ODEsystem \ xfList \ \text{with} \ (\lambda \ s. \ G \ s \wedge C \ s)$
using ** guarDiffEqtn-def* **by** *blast*
next
fix $a \ b$ **assume** $(a, b) \in ODEsystem \ xfList \ \text{with} \ (\lambda \ s. \ G \ s \wedge C \ s)$
then show $(a, b) \in ODEsystem \ xfList \ \text{with} \ G$
unfolding *guarDiffEqtn-def* **by**(*clarsimp, rule-tac x=t in exI, rule-tac x= φ_S in exI, simp*)
qed

1.4.3 "Solve Differential Equation"

lemma *prelim-dSolve*:

assumes *solHyp*: $(\lambda t. \ sol \ s[xfList \leftarrow uInput] \ t) \ solvesTheStoreIVP \ xfList \ \text{withInitState} \ s$
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 $\forall \ c. (s, c) \in (ODEsystem \ xfList \ \text{with} \ G) \longrightarrow Q \ c$
proof(*clarify*)
fix c **assume** $(s, c) \in (ODEsystem \ xfList \ \text{with} \ G)$

from this obtain $t::\text{real}$ **and** $\varphi_S::\text{real} \Rightarrow \text{real store}$
where $FHyp:t \geq 0 \wedge \varphi_S \ t = c \wedge \text{solvesStoreIVP } \varphi_S \ xfList \ s \wedge (\forall \ r \in \{0..t\}. G$
 $(\varphi_S \ r))$
using *guarDiffEqtn-def* **by** *auto*
from this and *uniqHyp* **have** $(\text{sol } s[xfList \leftarrow uInput] \ t) = \varphi_S \ t$ **by** *blast*
then have $cHyp:c = (\text{sol } s[xfList \leftarrow uInput] \ t)$ **using** *FHyp* **by** *simp*
from this have $G \ (\text{sol } s[xfList \leftarrow uInput] \ t)$ **using** *FHyp* **by** *force*
then show $Q \ c$ **using** *diffAssgn FHyp cHyp* **by** *auto*
qed

theorem dS:
assumes $\text{solHyp}:\forall \ s. \text{solvesStoreIVP } (\lambda t. \text{sol } s[xfList \leftarrow uInput] \ t) \ xfList \ s$
and $\text{uniqHyp}:\forall \ s \ X. \text{solvesStoreIVP } X \ xfList \ s \longrightarrow (\forall \ t \geq 0. (\text{sol } s[xfList \leftarrow uInput] \ t) = X \ t)$
shows $\text{wp } (\text{ODEsystem } xfList \text{ with } G) \ [Q] =$
 $[\lambda \ s. \forall \ t \geq 0. (\forall \ r \in \{0..t\}. G \ (\text{sol } s[xfList \leftarrow uInput] \ r)) \longrightarrow Q \ (\text{sol } s[xfList \leftarrow uInput] \ t)]$
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 ← 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 ← 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 $\text{solHyp}:\forall \ s. \text{solvesStoreIVP } (\lambda t. \text{sol } s[xfList \leftarrow uInput] \ t) \ xfList \ s$
and $\text{uniqHyp}:\forall \ s. \forall \ X. \text{solvesStoreIVP } X \ xfList \ s \longrightarrow (\forall \ t \geq 0. (\text{sol } s[xfList \leftarrow uInput] \ t) = X \ t)$
and $\text{diffAssgn}:\forall \ s. P \ s \longrightarrow (\forall \ t \geq 0. G \ (\text{sol } s[xfList \leftarrow uInput] \ t) \longrightarrow Q \ (\text{sol } s[xfList \leftarrow uInput] \ t))$
shows $\text{PRE } P \ (\text{ODEsystem } xfList \text{ with } G) \ \text{POST } Q$
apply(*clarsimp, subgoal-tac a=b*)
apply(*clarify, subst boxProgrPred-chrcrtrzn*)
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 $\text{funcsHyp}:\forall \ s \ g. \forall \ xf \in \text{set } xfList. \pi_2 \ xf \ (\text{override-on } s \ g \ \text{varDiffs}) = \pi_2 \ xf$
 s
and $\text{distinctHyp}:\text{distinct } (\text{map } \pi_1 \ xfList)$
and $\text{varsHyp}:\forall \ xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$

and $\text{lengthHyp}:\text{length } \text{xfList} = \text{length } \text{uInput}$
and $\text{solHyp1}:\forall \text{ uxf} \in \text{set } (\text{uInput} \otimes \text{xfList}). (\pi_1 \text{ uxf}) \ 0 \ (\text{sol } s) = (\text{sol } s) \ (\pi_1 \ (\pi_2 \text{ uxf}))$
and $\text{solHyp2}:\forall t \geq 0. ((\lambda \tau. (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau) \ x) \text{ has-vderiv-on } (\lambda \tau. f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau))) \ \{0..t\}$
and $\text{xfHyp}:(x, f) \in \text{set } \text{xfList} \text{ and } t\text{Hyp}:t \geq 0$
shows $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ (\partial \ x) = f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t)$
proof–
from xfHyp **obtain** u **where** $\text{xfuHyp}:(u, x, f) \in \text{set } (\text{uInput} \otimes \text{xfList})$
by $(\text{metis in-set-impl-in-set-zip2 lengthHyp})$
show $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ (\partial \ x) = f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t)$
proof($\text{cases } t=0$)
case True
have $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ 0) \ (\partial \ x) = f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ 0)$
using assms **and** $\text{to-sol-zero-its-dvars}$ **by** blast
then show $?thesis$ **using** True **by** blast
next
case False
from this **have** $t > 0$ **using** $t\text{Hyp}$ **by** simp
hence $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ (\partial \ x) = \text{vderiv-of } (\lambda r. u \ r \ (\text{sol } s)) \ \{0 < .. < (2 *_{\text{R}} t)\} \ t$
using $\text{xfuHyp assms to-sol-greater-than-zero-its-dvars}$ **by** blast
also have $\text{vderiv-of } (\lambda r. u \ r \ (\text{sol } s)) \ \{0 < .. < (2 *_{\text{R}} t)\} \ t = f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t)$
using $\text{assms xfuHyp } \langle t > 0 \rangle$ **and** $\text{vderiv-of-to-sol-its-vars}$ **by** blast
ultimately show $?thesis$ **by** simp
qed
qed

lemma conds4vdiffs :

assumes $\text{funcsHyp}:\forall s \ g. \forall \text{ xf} \in \text{set } \text{xfList}. \pi_2 \text{ xf} \ (\text{override-on } s \ g \ \text{varDiffs}) = \pi_2 \text{ xf } s$
and $\text{distinctHyp}:\text{distinct } (\text{map } \pi_1 \ \text{xfList})$
and $\text{varsHyp}:\forall \text{ xf} \in \text{set } \text{xfList}. \pi_1 \text{ xf} \notin \text{varDiffs}$
and $\text{lengthHyp}:\text{length } \text{xfList} = \text{length } \text{uInput}$
and $\text{solHyp1}:\forall \text{ uxf} \in \text{set } (\text{uInput} \otimes \text{xfList}). (\pi_1 \text{ uxf}) \ 0 \ (\text{sol } s) = (\text{sol } s) \ (\pi_1 \ (\pi_2 \text{ uxf}))$
and $\text{solHyp2}:\forall t \geq 0. \forall \text{ xf} \in \text{set } \text{xfList}. ((\lambda \tau. (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau) \ (\pi_1 \text{ xf})) \text{ has-vderiv-on } (\lambda \tau. (\pi_2 \text{ xf}) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau))) \ \{0..t\}$
shows $\forall t \geq 0. \forall \text{ xf} \in \text{set } \text{xfList}. (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ (\partial \ (\pi_1 \text{ xf})) = (\pi_2 \text{ xf}) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t)$
apply($\text{rule allI, rule impI, rule ballI, rule conds4vdiffs-prelim}$)
using assms **by** simp-all

lemma conds4Consts :

assumes $\text{varsHyp}:\forall \text{ xf} \in \text{set } \text{xfList}. \pi_1 \text{ xf} \notin \text{varDiffs}$
shows $\forall x. x \notin (\pi_1 \ (\text{set } \text{xfList})) \longrightarrow (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ (\partial \ x) = 0$
using varsHyp **apply**($\text{induct } \text{xfList } \text{uInput rule: list-induct2'}$)
apply($\text{simp-all add: override-on-def varDiffs-def vdiff-def}$)

by *clarsimp*

lemma *conds4InitState*:

assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)

and *lengthHyp*:*length* *xfList* = *length* *uInput*

and *varsHyp*: \forall *xf* \in *set* *xfList*. π_1 *xf* \notin *varDiffs*

and *solHyp1*: \forall *uxf* \in *set* (*uInput* \otimes *xfList*). (π_1 *uxf*) 0 (*sol* *s*) = (*sol* *s*) (π_1 (π_2 *uxf*))

and *xfHyp*:(*x*, *f*) \in *set* *xfList*

shows (*sol* *s*[*xfList* \leftarrow *uInput*] 0) *x* = *s* *x*

proof–

from *xfHyp* **obtain** *u* **where** *uxfHyp*:(*u*, *x*, *f*) \in *set* (*uInput* \otimes *xfList*)

by (*metis in-set-impl-in-set-zip2 lengthHyp*)

from *varsHyp* **have** *toZeroHyp*:(*sol* *s*) *x* = *s* *x* **using** *override-on-def xfHyp* **by** *auto*

from *uxfHyp* **and** *solHyp1* **have** *u* 0 (*sol* *s*) = (*sol* *s*) *x* **by** *fastforce*

also **have** (*sol* *s*[*xfList* \leftarrow *uInput*] 0) *x* = *u* 0 (*sol* *s*)

using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*

ultimately show (*sol* *s*[*xfList* \leftarrow *uInput*] 0) *x* = *s* *x* **using** *toZeroHyp* **by** *simp*

qed

lemma *conds4RestOfStrings*:

assumes *x* \notin (π_1 (*set* *xfList*)) \cup *varDiffs*

shows (*sol* *s*[*xfList* \leftarrow *uInput*] *t*) *x* = *s* *x*

using *assms* **apply**(*induct xfList uInput rule: list-induct2'*)

by(*auto simp: varDiffs-def*)

lemma *conds4storeIVP-on-toSol*:

assumes *funcsHyp*: \forall *s g*. \forall *xf* \in *set* *xfList*. π_2 *xf* (*override-on* *s g* *varDiffs*) = π_2 *xf* *s*

and *distinctHyp*:*distinct* (*map* π_1 *xfList*)

and *lengthHyp*:*length* *xfList* = *length* *uInput*

and *varsHyp*: \forall *xf* \in *set* *xfList*. π_1 *xf* \notin *varDiffs*

and *solHyp1*: \forall *uxf* \in *set* (*uInput* \otimes *xfList*). (π_1 *uxf*) 0 (*sol* *s*) = (*sol* *s*) (π_1 (π_2 *uxf*))

and *solHyp2*: \forall *t* \geq 0. \forall *xf* \in *set* *xfList*.

((λt . (*sol* *s*[*xfList* \leftarrow *uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λt . π_2 *xf* (*sol* *s*[*xfList* \leftarrow *uInput*] *t*))) {0..*t*})

shows *solvesStoreIVP* (λt . (*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*: $\forall s\ g. \forall xf \in \text{set } xfList. \pi_2\ xf\ (\text{override-on } s\ g\ \text{varDiffs}) = \pi_2\ xf\ s$
and *distinctHyp*:*distinct* (*map* $\pi_1\ xfList$)
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1\ xf \notin \text{varDiffs}$
and *solHyp1*: $\forall s. \forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$
and *solHyp2*: $\forall s. \forall t \geq 0. \forall xf \in \text{set } xfList.$
 $((\lambda t. (sol\ s[xfList \leftarrow uInput]\ t)\ (\pi_1\ xf))\ \text{has-vderiv-on}\ (\lambda t. \pi_2\ xf\ (sol\ s[xfList \leftarrow uInput]\ t)))\ \{0..t\}$
and *uniqHyp*: $\forall s. \forall X. \text{solvesStoreIVP } X\ xfList\ s \longrightarrow (\forall t \geq 0. (sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$
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* ($\{0..t\} \times UNIV$) $(\lambda(t, (r::real)). f\ (\varphi_s\ t))$
shows *unique-on-bounded-closed* $0\ \{0..t\}\ \tau\ (\lambda t\ r. f\ (\varphi_s\ t))\ UNIV$ (*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* $\varphi_s\ xfList\ a$
shows $\varphi_s\ 0 = \text{override-on } a\ (\varphi_s\ 0)\ \text{varDiffs}$
apply(*rule ext, subgoal-tac* $x \notin \text{varDiffs} \longrightarrow \varphi_s\ 0\ x = a\ x$)
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*: $\forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$
 $(\lambda(t, (r::real)). (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ t))$
and *eqDerivs*: $\forall xf \in \text{set } xfList. \forall \tau \in \{0..t\}. (\pi_2\ xf)\ (\varphi_s\ \tau) = (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau)$
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  $(cases\ x \in (\pi_1(\downarrow set\ xfList)) \cup varDiffs)$ 
  case False
    then have  $notInVars:x \notin (\pi_1(\downarrow set\ xfList)) \cup varDiffs$  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  $x \in (\pi_1(\downarrow set\ xfList)) \vee x \in varDiffs$  by simp
    from this show ?thesis
  proof
    assume  $x \in (\pi_1(\downarrow set\ xfList))$ 
    from this obtain  $f$  where  $xfHyp:(x, f) \in set\ xfList$  by fastforce

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

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

    from solHyp and xfHyp have  $2:((\lambda\tau. (sol\ s[xfList \leftarrow uInput]\ \tau)\ x)\ solves\ ode$ 
       $(\lambda\tau\ r. f\ (sol\ s[xfList \leftarrow uInput]\ \tau)))\{0..t\}\ UNIV \wedge (sol\ s[xfList \leftarrow uInput]\ 0)$ 
       $x = s\ x$ 
    using solvesStoreIVP-def tHyp by fastforce

    from tHyp and contHyp have  $\forall\ xf \in set\ xfList. unique\ on\ bounded\ closed\ 0$ 
       $\{0..t\}\ (s\ (\pi_1\ xf))$ 
       $(\lambda\tau\ r. (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau))\ UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$ 

    apply (clarify) apply (rule conds4UniqSol) by (auto)
    from this have  $3:unique\ on\ bounded\ closed\ 0\ \{0..t\}\ (s\ x)\ (\lambda\tau\ r. f\ (sol$ 
       $s[xfList \leftarrow uInput]\ \tau))$ 
       $UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$  using xfHyp by fastforce
    from 1 2 and 3 show  $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x$ 
    using unique-on-bounded-closed.unique-solution using real-Icc-closed-segment

```



```

tHyp by blast
  next
    assume  $x \in \text{varDiffs}$ 
    then obtain  $y$  where  $xDef: x = \partial y$  by (auto simp: varDiffs-def)
    show ( $\text{sol } s[xfList \leftarrow uInput] \ t$ )  $x = \varphi_s \ t \ x$ 
    proof( $\text{cases } y \in \text{set } (\text{map } \pi_1 \ xfList)$ )
    case True
      then obtain  $f$  where  $xfHyp: (y, f) \in \text{set } xfList$  by fastforce
      from tHyp and Fsolves have  $\varphi_s \ t \ x = f \ (\varphi_s \ t)$ 
      using solves-store-ivpD(3) xfHyp xDef by force
      also have ( $\text{sol } s[xfList \leftarrow uInput] \ t$ )  $x = f \ (\text{sol } s[xfList \leftarrow uInput] \ t)$ 
      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  $\varphi_s \ t \ x = 0$ 
      using xDef solves-store-ivpD(2) Fsolves tHyp by simp
      also have ( $\text{sol } s[xfList \leftarrow uInput] \ t$ )  $x = 0$ 
      using False solHyp tHyp solves-store-ivpD(2) xDef by fastforce
      ultimately show ?thesis by simp
    qed
  qed
qed
qed

theorem dSolveUBC:
assumes contHyp:  $\forall s. \forall t \geq 0. \forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$ 

  ( $\lambda(t, (r::real)). (\pi_2 \ xf) \ (\text{sol } s[xfList \leftarrow uInput] \ t)$ )
and solHyp:  $\forall s. \text{solvesStoreIVP } \varphi_s \ xfList \ s$  and  $(\lambda t. (\text{sol } s[xfList \leftarrow uInput] \ t)) \ xfList \ s$ 
and uniqHyp:  $\forall s. \forall \varphi_s. \varphi_s \ \text{solvesTheStoreIVP } xfList \ \text{withInitState } s \longrightarrow$ 
  ( $\forall t \geq 0. \forall xf \in \text{set } xfList. \forall r \in \{0..t\}. (\pi_2 \ xf) \ (\varphi_s \ r) = (\pi_2 \ xf) \ (\text{sol } s[xfList \leftarrow uInput] \ r)$ )
and diffAssgn:  $\forall s. P \ s \longrightarrow (\forall t \geq 0. G \ (\text{sol } s[xfList \leftarrow uInput] \ t) \longrightarrow Q \ (\text{sol } s[xfList \leftarrow uInput] \ t))$ 
shows PRE  $P \ (ODEsystem \ xfList \ \text{with } G) \ POST \ Q$ 
apply(rule-tac uInput=uInput in dSolve)
prefer 2 subgoal proof(clarify)
fix  $s::real \ \text{store}$  and  $\varphi_s::real \Rightarrow real \ \text{store}$  and  $t::real$ 
assume isSol:solvesStoreIVP  $\varphi_s \ xfList \ s$  and  $sHyp: 0 \leq t$ 
from this and uniqHyp have  $\forall xf \in \text{set } xfList. \forall t \in \{0..t\}. (\pi_2 \ xf) \ (\varphi_s \ t) = (\pi_2 \ xf) \ (\text{sol } s[xfList \leftarrow uInput] \ t)$  by auto
also have  $\forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$ 
  ( $\lambda(t, (r::real)). (\pi_2 \ xf) \ (\text{sol } s[xfList \leftarrow uInput] \ t)$ ) using contHyp sHyp by blast
ultimately show ( $\text{sol } s[xfList \leftarrow uInput] \ t$ )  $= \varphi_s \ t$ 
using sHyp isSol ubcStoreUniqueSol solHyp by simp
qed using assms by simp-all

```

```

theorem dSolve-toSolveUBC:
assumes funcsHyp:  $\forall s \ g. \forall xf \in \text{set } xfList. \pi_2 \ xf \ (\text{override-on } s \ g \ \text{varDiffs}) = \pi_2 \ xf$ 

```

s
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *varsHyp*: $\forall x f \in \text{set } xfList. \pi_1 x f \notin \text{varDiffs}$
and *solHyp1*: $\forall s. \forall u x f \in \text{set } (uInput \otimes xfList). \pi_1 u x f 0 (sol s) = sol s (\pi_1 (\pi_2 u x f))$
and *solHyp2*: $\forall s. \forall t \geq 0. \forall x f \in \text{set } xfList. ((\lambda t. (sol s[xfList \leftarrow uInput] t) (\pi_1 x f))$
has-vderiv-on
 $(\lambda t. \pi_2 x f (sol s[xfList \leftarrow uInput] t))) \{0..t\}$
and *contHyp*: $\forall s. \forall t \geq 0. \forall x f \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$
 $(\lambda(t, (r::real)). (\pi_2 x f) (sol s[xfList \leftarrow uInput] t))$
and *uniqHyp*: $\forall s. \forall \varphi_s. \varphi_s \text{ solvesTheStoreIVP } xfList \text{ withInitState } s \longrightarrow$
 $(\forall t \geq 0. \forall x f \in \text{set } xfList. \forall r \in \{0..t\}. (\pi_2 x f) (\varphi_s r) = (\pi_2 x f) (sol s[xfList \leftarrow uInput]$
 $r))$
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*

1.4.4 "Differential Invariant."

lemma *solvesStoreIVP-couldBeModified*:
fixes *F*:*real* \Rightarrow *real store*
assumes *vars*: $\forall t \geq 0. \forall x f \in \text{set } xfList. ((\lambda t. F t (\pi_1 x f)) \text{ solves-ode } (\lambda t r. \pi_2 x f (F t))) \{0..t\} UNIV$
and *dvars*: $\forall t \geq 0. \forall x f \in \text{set } xfList. (F t (\partial (\pi_1 x f))) = (\pi_2 x f) (F t)$
shows $\forall t \geq 0. \forall r \in \{0..t\}. \forall x f \in \text{set } xfList.$
 $((\lambda t. F t (\pi_1 x f)) \text{ has-vector-derivative } F r (\partial (\pi_1 x f))) (at r \text{ within } \{0..t\})$
proof(*clarify*, *rename-tac* *t r x f*)
fix *x f* **and** *t r*:*real*
assume *tHyp*: $0 \leq t$ **and** *xfHyp*: $(x, f) \in \text{set } xfList$ **and** *rHyp*: $r \in \{0..t\}$
from *this* **and** *vars* **have** $((\lambda t. F t x) \text{ solves-ode } (\lambda t r. f (F t))) \{0..t\} UNIV$
using *tHyp* **by** *fastforce*
hence $\ast: \forall r \in \{0..t\}. ((\lambda t. F t x) \text{ has-vector-derivative } (\lambda t. f (F t)) r) (at r \text{ within } \{0..t\})$
by (*simp add*: *solves-ode-def* *has-vderiv-on-def* *tHyp*)
have $\forall t \geq 0. \forall r \in \{0..t\}. \forall x f \in \text{set } xfList. (F r (\partial (\pi_1 x f))) = (\pi_2 x f) (F r)$
using *assms* **by** *auto*
from *this* *rHyp* **and** *xfHyp* **have** $(F r (\partial x)) = f (F r)$ **by** *force*
then show $((\lambda t. F t (\pi_1 (x, f))) \text{ has-vector-derivative } F r (\partial (\pi_1 (x, f)))) (at r \text{ within } \{0..t\})$
using \ast *rHyp* **by** *auto*
qed

lemma *derivationLemma-baseCase*:
fixes *F*:*real* \Rightarrow *real store*
assumes *solves*:*solvesStoreIVP* *F* *xfList* *a*

shows $\forall x \in (UNIV - \text{varDiffs}). \forall t \geq 0. \forall r \in \{0..t\}.$
 $((\lambda t. F t x) \text{ has-vector-derivative } F r (\partial x)) \text{ (at } r \text{ within } \{0..t\})$
proof
fix x
assume $x \in UNIV - \text{varDiffs}$
then have $\text{notVarDiff}:\forall z. x \neq \partial z$ **using** varDiffs-def **by** fastforce
show $\forall t \geq 0. \forall r \in \{0..t\}. ((\lambda t. F t x) \text{ has-vector-derivative } F r (\partial x)) \text{ (at } r \text{ within } \{0..t\})$
proof($\text{cases } x \in \text{set (map } \pi_1 \text{ xfList)}$)
case True
from this and solves have $\forall t \geq 0. \forall r \in \{0..t\}. \forall xf \in \text{set xfList}.$
 $((\lambda t. F t (\pi_1 xf)) \text{ has-vector-derivative } F r (\partial (\pi_1 xf))) \text{ (at } r \text{ within } \{0..t\})$
apply($\text{rule-tac solvesStoreIVP-couldBeModified}$) **using** $\text{solves solves-store-ivpD}$
by auto
from this show $?thesis$ **using** True **by** auto
next
case False
from this notVarDiff and solves have $\text{const}:\forall t \geq 0. F t x = a x$
using $\text{solves-store-ivpD(1)}$ **by** ($\text{simp add: varDiffs-def}$)
have $\text{constD}:\forall t \geq 0. \forall r \in \{0..t\}. ((\lambda r. a x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$
by ($\text{auto intro: derivative-eq-intros}$)
{fix $t r::\text{real}$
assume $t \geq 0$ **and** $r \in \{0..t\}$
hence $((\lambda s. a x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$ **by** (simp add: constD)
moreover have $\bigwedge s. s \in \{0..t\} \implies (\lambda r. F r x) s = (\lambda r. a x) s$
using const **by** ($\text{simp add: } \langle 0 \leq t \rangle$)
ultimately have $((\lambda s. F s x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$
using $\text{has-vector-derivative-transform}$ **by** ($\text{metis } \langle r \in \{0..t\} \rangle$)
hence $\text{isZero}:\forall t \geq 0. \forall r \in \{0..t\}. ((\lambda t. F t x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$ **by** blast
from $\text{False solves and notVarDiff have}$ $\forall t \geq 0. F t (\partial x) = 0$
using $\text{solves-store-ivpD(2)}$ **by** simp
then show $?thesis$ **using** isZero **by** simp
qed
qed

lemma $\text{derivationLemma}:$
assumes $\text{solvesStoreIVP } F \text{ xfList } a$
and $t\text{Hyp}:t \geq 0$
and $\text{termVarsHyp}:\forall x \in \text{trmVars } \eta. x \in (UNIV - \text{varDiffs})$
shows $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F r)) \text{ (at } r \text{ within } \{0..t\})$
using termVarsHyp **proof**($\text{induction } \eta$)
case ($\text{Const } r$)
then show $?case$ **by** simp
next
case ($\text{Var } y$)

```

then have  $yHyp: y \in UNIV - varDiffs$  by auto
from this  $tHyp$  and  $assms(1)$  show  $?case$ 
using  $derivationLemma-baseCase$  by auto
next
case ( $Mns \ \eta$ )
then show  $?case$ 
apply( $clarsimp$ )
by( $rule \ derivative-intros, simp$ )
next
case ( $Sum \ \eta1 \ \eta2$ )
then show  $?case$ 
apply( $clarsimp$ )
by( $rule \ derivative-intros, simp-all$ )
next
case ( $Mult \ \eta1 \ \eta2$ )
then show  $?case$ 
apply( $clarsimp$ )
apply( $subgoal-tac \ ((\lambda s. \llbracket \eta1 \rrbracket_t (F \ s) *_{\mathbb{R}} \llbracket \eta2 \rrbracket_t (F \ s)) \text{ has-vector-derivative } \llbracket \partial_t \ \eta1 \rrbracket_t (F \ r) \cdot \llbracket \eta2 \rrbracket_t (F \ r) + \llbracket \eta1 \rrbracket_t (F \ r) \cdot \llbracket \partial_t \ \eta2 \rrbracket_t (F \ r)) \text{ (at } r \text{ within } \{0..t\}, simp)$ )
apply( $rule-tac \ f'1 = \llbracket \partial_t \ \eta1 \rrbracket_t (F \ r)$  and  $g'1 = \llbracket \partial_t \ \eta2 \rrbracket_t (F \ r)$  in  $derivative-eq-intros(25)$ )
by ( $simp-all \ add: \text{has-field-derivative-iff-has-vector-derivative}$ )
qed

```

```

lemma  $diff-subst-prppty-4terms$ :
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  $\llbracket \partial_t \ \eta \rrbracket_t (F \ t) = \llbracket (map \ (vdiff \circ \pi_1) \ xfList) \otimes uInput \rrbracket_t \langle \partial_t \ \eta \rangle_t (F \ t)$ 
using  $termVarsHyp$  apply( $induction \ \eta$ ) apply( $simp-all \ add: substList-help2$ )
using  $listsHyp$  and  $solves$  apply( $induct \ xfList \ uInput \ rule: list-induct2', simp, simp, simp$ )
proof( $clarify, rename-tac \ y \ g \ xfTail \ \vartheta \ trmTail \ x$ )
fix  $x \ y::string$  and  $\vartheta::trms$  and  $g$  and  $xfTail::(string \times (real \ store \Rightarrow real)) \ list$ 
and  $trmTail$ 
assume  $IH: \bigwedge x. x \notin varDiffs \Longrightarrow map \ \pi_2 \ xfTail = map \ tval \ trmTail \Longrightarrow$ 
 $\forall \ xf \in set \ xfTail. \ F \ t \ (\partial \ (\pi_1 \ xf)) = \pi_2 \ xf \ (F \ t) \Longrightarrow$ 
 $F \ t \ (\partial \ x) = \llbracket (map \ (vdiff \circ \pi_1) \ xfTail) \otimes trmTail \rrbracket_t \langle t_V \ (\partial \ x) \rangle_t (F \ t)$ 
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  $*$ :  $\llbracket (map \ (vdiff \circ \pi_1) \ xfTail) \otimes trmTail \rrbracket_t \langle Var \ (\partial \ x) \rangle_t (F \ t) = F \ t \ (\partial \ x)$ 
using  $tHyp$  by auto
show  $F \ t \ (\partial \ x) = \llbracket (map \ (vdiff \circ \pi_1) \ ((y, g) \# xfTail)) \otimes (\vartheta \# trmTail) \rrbracket_t \langle t_V \ (\partial \ x) \rangle_t (F \ t)$ 
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  $((\text{map } (\text{vdiff} \circ \pi_1) ((y, g) \# \text{xfTail})) \otimes (\vartheta \# \text{trmTail})) \langle t_V$ 
 $(\partial x) \rangle = \vartheta$  by simp
  also from 3 tHyp have  $F t (\partial y) = g (F t)$  by simp
  moreover from 2 have  $\llbracket \vartheta \rrbracket_t (F t) = g (F t)$  by simp
  ultimately have ?thesis by (simp add:  $\langle x = y \rangle$ )}
moreover
{assume  $x \neq y \wedge x \in \text{set } (\text{map } \pi_1 \text{xfTail})$ 
  then have  $\partial x \neq \partial y$  using vdiff-inj by auto
  from this have  $((\text{map } (\text{vdiff} \circ \pi_1) ((y, g) \# \text{xfTail})) \otimes (\vartheta \# \text{trmTail})) \langle t_V$ 
 $(\partial x) \rangle =$ 
 $((\text{map } (\text{vdiff} \circ \pi_1) \text{xfTail}) \otimes \text{trmTail}) \langle t_V (\partial x) \rangle$  by simp
  hence ?thesis using * by simp}
ultimately show ?thesis by blast
next
case False
  then have  $((\text{map } (\text{vdiff} \circ \pi_1) ((y, g) \# \text{xfTail})) \otimes (\vartheta \# \text{trmTail})) \langle t_V$ 
 $(\partial x) \rangle =$ 
 $t_V (\partial x)$ 
  using substList-cross-vdiff-on-non-occurring-var by (metis (no-types, lifting) List.map.compositionality)
  thus ?thesis by simp
qed
qed

```

lemma *eqInVars-impl-eqInTrms*:
assumes *termVarsHyp*: $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$
and *initHyp*: $\forall x. x \notin \text{varDiffs} \longrightarrow b \ x = a \ x$
shows $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t b$
using *assms* by (induction η , simp-all)

lemma *non-empty-funList-implies-non-empty-trmList*:
shows $\forall \text{list}. (x, f) \in \text{set list} \wedge \text{map } \pi_2 \text{ list} = \text{map tval tList} \longrightarrow (\exists \vartheta. \llbracket \vartheta \rrbracket_t = f \wedge$
 $\vartheta \in \text{set tList})$
by (induction tList, auto)

lemma *dInvForTrms-prelim*:
assumes *substHyp*:
 $\forall st. G \ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{xfList}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t st = 0$
and *termVarsHyp*: $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$
and *listsHyp*: $\text{map } \pi_2 \text{xfList} = \text{map tval uInput}$
shows $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem } \text{xfList} \text{ with } G)) \longrightarrow \llbracket \eta \rrbracket_t c = 0$
proof (clarify)
fix *c* **assume** *aHyp*: $\llbracket \eta \rrbracket_t a = 0$ **and** *cHyp*: $(a, c) \in \text{ODEsystem } \text{xfList} \text{ with } G$
from this obtain *t*: *real* **and** *F*: *real* \Rightarrow *real store*
where *tcHyp*: $t \geq 0 \wedge F \ t = c \wedge \text{solvesStoreIVP } F \ \text{xfList } a \wedge (\forall r \in \{0..t\}. G \ (F \ r))$

using *guarDiffEqtn-def* **by** auto
then have $\forall x. x \notin \text{varDiffs} \longrightarrow F \ 0 \ x = a \ x$ **using** *solves-store-ivpD(6)* **by** blast
from this have $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t (F \ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms*

by *blast*
 hence $\text{obs1}:\llbracket \eta \rrbracket_t (F\ 0) = 0$ **using** *aHyp* **by** *simp*
 from *tcHyp* **have** $\text{obs2}:\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F\ r))$ (at *r* within $\{0..t\}$) **using** *derivationLemma termVarsHyp* **by** *blast*
have $\forall r \in \{0..t\}. \forall xf \in \text{set } xfList. F\ r\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ r)$
using *tcHyp solves-store-ivpD(3)* **by** *fastforce*
 hence $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F\ r) = \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F\ r)$
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r \in \{0..t\}. \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F\ r) = 0$
using *solves-store-ivpD(2) tcHyp* **by** *fastforce*
ultimately have $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s)) \text{ has-vector-derivative } 0)$ (at *r* within $\{0..t\}$)
using *obs2* **by** *auto*
 from *this* and *tcHyp* **have** $\forall s \in \{0..t\}. ((\lambda x. \llbracket \eta \rrbracket_t (F\ x)) \text{ has-derivative } (\lambda x. x *_R 0))$
 (at *s* within $\{0..t\}$) **by** (*metis has-vector-derivative-def*)
 hence $\llbracket \eta \rrbracket_t (F\ t) - \llbracket \eta \rrbracket_t (F\ 0) = (\lambda x. x *_R 0) (t - 0)$
using *mvt-very-simple* and *tcHyp* **by** *fastforce*
then show $\llbracket \eta \rrbracket_t c = 0$ **using** *obs1 tcHyp* **by** *auto*
qed

theorem *dInvForTrms*:

assumes $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
 $\llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t st = 0$
and *termVarsHyp:trmVars* $\eta \subseteq (UNIV - \text{varDiffs})$
and *listsHyp:map* $\pi_2\ xfList = \text{map } \text{tval } uInput$
and *eta-f:f* $= \llbracket \eta \rrbracket_t$
shows $PRED\ (\lambda s. f\ s = 0)\ (ODEsystem\ xfList\ \text{with } G)\ POST\ (\lambda s. f\ s = 0)$
using *eta-f proof(clarsimp)*
fix *a b*
assume $(a, b) \in [\lambda s. \llbracket \eta \rrbracket_t s = 0]$ **and** $f = \llbracket \eta \rrbracket_t$
from *this* **have** *aHyp:a = b* $\wedge \llbracket \eta \rrbracket_t a = 0$ **by** (*metis (full-types) d-p2r rdom-p2r-contents*)
have $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ \text{with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0)$
using *assms dInvForTrms-prelim* **by** *metis*
from *this* and *aHyp* **have** $\forall c. (a, c) \in (ODEsystem\ xfList\ \text{with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0$ **by** *blast*
thus $(a, b) \in wp\ (ODEsystem\ xfList\ \text{with } G)\ [\lambda s. \llbracket \eta \rrbracket_t s = 0]$
using *aHyp* **by** (*simp add: boxProgrPred-chrctrztn*)
qed

lemma *diff-subst-prprty-4props*:

assumes *solves*: $\forall xf \in \text{set } xfList. F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$
and *tHyp:t* ≥ 0
and *listsHyp:map* $\pi_2\ xfList = \text{map } \text{tval } uInput$
and *propVarsHyp:propVars* $\varphi \subseteq (UNIV - \text{varDiffs})$
shows $\llbracket \partial_P \varphi \rrbracket_P (F\ t) = \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P \varphi \rrbracket_P (F\ t)$
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:
 $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket set\ xfList \rrbracket)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$ 
 $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t\ st \geq 0$ 
and termVarsHyp:  $trmVars\ \eta \subseteq (UNIV - varDiffs)$ 
and listsHyp:  $map\ \pi_2\ xfList = map\ tval\ uInput$ 
shows  $\llbracket \eta \rrbracket_t\ a > 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t\ c > 0)$ 
and  $\llbracket \eta \rrbracket_t\ a \geq 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t\ c \geq 0)$ 
proof(clarify)
fix c assume aHyp:  $\llbracket \eta \rrbracket_t\ a > 0$  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  $\llbracket \eta \rrbracket_t\ a = \llbracket \eta \rrbracket_t\ (F\ 0)$  using termVarsHyp eqInVars-impl-eqInTrms
by blast
hence obs1:  $\llbracket \eta \rrbracket_t\ (F\ 0) > 0$  using aHyp tcHyp by simp
from tcHyp have obs2:  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t\ (F\ s))\ has\_vector\_derivative\ \llbracket \partial_t \eta \rrbracket_t\ (F\ r))\ (at\ r\ within\ \{0..t\})$  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  $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t\ (F\ r) = \llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t\ (F\ r)$ 
using diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp by fastforce
also from substHyp have  $\forall r \in \{0..t\}. \llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t\ (F\ r) \geq 0$ 
using solves-store-ivpD(2) tcHyp by (metis atLeastAtMost-iff)
ultimately have  $*: \forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t\ (F\ r) \geq 0$  by (simp)
from obs2 and tcHyp have  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t\ (F\ s))\ has\_derivative\ (\lambda x. x *_R (\llbracket \partial_t \eta \rrbracket_t\ (F\ r))))\ (at\ r\ within\ \{0..t\})$  by (simp add: has-vector-derivative-def)

hence  $\exists r \in \{0..t\}. \llbracket \eta \rrbracket_t\ (F\ t) - \llbracket \eta \rrbracket_t\ (F\ 0) = t \cdot ((\llbracket \partial_t \eta \rrbracket_t) (F\ r))$ 
using mvt-very-simple and tcHyp by fastforce
then obtain r where  $\llbracket \partial_t \eta \rrbracket_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge \llbracket \partial_t \eta \rrbracket_t\ (F\ t) \geq 0$ 
 $\wedge \llbracket \eta \rrbracket_t\ (F\ t) - \llbracket \eta \rrbracket_t\ (F\ 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t\ (F\ r))$ 
using  $*$  tcHyp by (meson atLeastAtMost-iff order-refl)
thus  $\llbracket \eta \rrbracket_t\ c > 0$ 
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  $0 \leq \llbracket \eta \rrbracket_t\ a \longrightarrow (\forall c. (a, c) \in ODEsystem\ xfList\ with\ G \longrightarrow 0 \leq \llbracket \eta \rrbracket_t\ c)$ 
proof(clarify)

```

fix c **assume** $aHyp:\llbracket \eta \rrbracket_t a \geq 0$ **and** $cHyp:(a, c) \in ODEsystem\ xfList$ **with** G
from this obtain $t::real$ **and** $F::real \Rightarrow real\ store$
where $tcHyp:t \geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a \wedge (\forall r \in \{0..t\}. G\ (F\ r))$

using *guarDiffEqtn-def* **by** *auto*
then have $\forall x. x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$ **using** *solves-store-ivpD(6)* **by** *blast*
from this have $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t (F\ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms*
by *blast*
hence $obs1:\llbracket \eta \rrbracket_t (F\ 0) \geq 0$ **using** $aHyp\ tcHyp$ **by** *simp*
from $tcHyp$ **have** $obs2:\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s))\ has-vector-derivative\ \llbracket \partial_t \eta \rrbracket_t (F\ r))\ (at\ r\ within\ \{0..t\})$ **using** *derivationLemma termVarsHyp* **by** *blast*
have $(\forall t \geq 0. \forall xf \in set\ xfList. F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using $tcHyp\ solves-store-ivpD(3)$ **by** *blast*
from this and $tcHyp$ **have** $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F\ r) =$
 $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t \eta \rangle \rrbracket_t (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from $substHyp$ **have** $\forall r \in \{0..t\}. \llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t \eta \rangle \rrbracket_t (F\ r) \geq 0$
using $solves-store-ivpD(2)\ tcHyp$ **by** *(metis atLeastAtMost-iff)*
ultimately have $*\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F\ r) \geq 0$ **by** *(simp)*
from $obs2$ **and** $tcHyp$ **have** $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s))\ has-derivative\ (\lambda x. x *_{\mathbb{R}} (\llbracket \partial_t \eta \rrbracket_t (F\ r))))\ (at\ r\ within\ \{0..t\})$ **by** *(simp add: has-vector-derivative-def)*

hence $\exists r \in \{0..t\}. \llbracket \eta \rrbracket_t (F\ t) - \llbracket \eta \rrbracket_t (F\ 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F\ r))$
using *mut-very-simple* **and** $tcHyp$ **by** *fastforce*
then obtain r **where** $\llbracket \partial_t \eta \rrbracket_t (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge \llbracket \partial_t \eta \rrbracket_t (F\ t) \geq 0$
 $\wedge \llbracket \eta \rrbracket_t (F\ t) - \llbracket \eta \rrbracket_t (F\ 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F\ r))$
using $*\ tcHyp$ **by** *(meson atLeastAtMost-iff order-refl)*
thus $\llbracket \eta \rrbracket_t c \geq 0$
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 $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ (\vartheta \prec \eta) \rrbracket_P\ st$
shows $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle \rrbracket_t\ st \geq 0$
using *assms* **by** *(auto)*

lemma *leq-pval-to-tval*:

assumes $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ (\vartheta \preceq \eta) \rrbracket_P\ st$
shows $\llbracket ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle \rrbracket_t\ st \geq 0$
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

$\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ } xfList) \otimes uInput) \upharpoonright \partial_P \varphi \rrbracket_P st$
and $\text{propVarsHyp}:\text{propVars } \varphi \subseteq (UNIV - \text{varDiffs})$
and $\text{listsHyp}:\text{map } \pi_2 \text{ } xfList = \text{map } \text{tval } uInput$
shows $\llbracket \varphi \rrbracket_P a \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem } xfList \text{ with } G) \longrightarrow \llbracket \varphi \rrbracket_P c)$
proof(*clarify*)
fix c **assume** $aHyp:\llbracket \varphi \rrbracket_P a$ **and** $cHyp:(a, c) \in \text{ODEsystem } xfList \text{ with } G$
from this obtain $t::\text{real}$ **and** $F::\text{real} \Rightarrow \text{real store}$
where $tcHyp:t \geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ } xfList \text{ } a$ **using** *guarDiffEqtn-def*
by *auto*
from $aHyp$ propVarsHyp **and** substHyp **show** $\llbracket \varphi \rrbracket_P c$
proof(*induction* φ)
case (*Eq* $\vartheta \eta$)
hence $\text{hyp}:\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ } xfList) \otimes uInput) \upharpoonright \partial_P (\vartheta \doteq \eta) \rrbracket_P st$ **by** *blast*
then have $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ } xfList) \otimes uInput) \langle \partial_t (\vartheta \oplus (\ominus \eta)) \rangle_t st = 0$ **by** *simp*
also have $\text{trmVars } (\vartheta \oplus (\ominus \eta)) \subseteq UNIV - \text{varDiffs}$ **using** *Eq.premis(2)* **by** *simp*
moreover have $\llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t a = 0$ **using** *Eq.premis(1)* **by** *simp*
ultimately have $(\forall c. (a, c) \in \text{ODEsystem } xfList \text{ with } G \longrightarrow \llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t c = 0)$
using *dInvForTrms-prelim listsHyp* **by** *blast*
hence $\llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t (F t) = 0$ **using** $tcHyp$ $cHyp$ **by** *simp*
from this have $\llbracket \vartheta \rrbracket_t (F t) = \llbracket \eta \rrbracket_t (F t)$ **by** *simp*
also have $(\llbracket \vartheta \doteq \eta \rrbracket_P) c = (\llbracket \vartheta \rrbracket_t (F t) = \llbracket \eta \rrbracket_t (F t))$ **using** $tcHyp$ **by** *simp*
ultimately show *?case* **by** *simp*
next
case (*Less* $\vartheta \eta$)
hence $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $0 \leq (\llbracket (\text{map } (vdiff \circ \pi_1) \text{ } xfList \otimes uInput) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle_t st$
using *less-pval-to-tval* **by** *metis*
also from *Less.premis(2)* **have** $\text{trmVars } (\eta \oplus (\ominus \vartheta)) \subseteq UNIV - \text{varDiffs}$ **by** *simp*
moreover have $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t a > 0$ **using** *Less.premis(1)* **by** *simp*
ultimately have $(\forall c. (a, c) \in \text{ODEsystem } xfList \text{ with } G \longrightarrow \llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t c > 0)$
using *dInvForProps-prelim(1) listsHyp* **by** *blast*
hence $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t (F t) > 0$ **using** $tcHyp$ $cHyp$ **by** *simp*
from this have $\llbracket \eta \rrbracket_t (F t) > \llbracket \vartheta \rrbracket_t (F t)$ **by** *simp*
also have $\llbracket \vartheta \prec \eta \rrbracket_P c = (\llbracket \vartheta \rrbracket_t (F t) < \llbracket \eta \rrbracket_t (F t))$ **using** $tcHyp$ **by** *simp*
ultimately show *?case* **by** *simp*
next
case (*Leq* $\vartheta \eta$)
hence $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $0 \leq (\llbracket (\text{map } (vdiff \circ \pi_1) \text{ } xfList \otimes uInput) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle_t st$ **using** *leq-pval-to-tval*
by *metis*
also from *Leq.premis(2)* **have** $\text{trmVars } (\eta \oplus (\ominus \vartheta)) \subseteq UNIV - \text{varDiffs}$ **by** *simp*
moreover have $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t a \geq 0$ **using** *Leq.premis(1)* **by** *simp*
ultimately have $(\forall c. (a, c) \in \text{ODEsystem } xfList \text{ with } G \longrightarrow \llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t c \geq 0)$
using *dInvForProps-prelim(2) listsHyp* **by** *blast*

hence $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t (F t) \geq 0$ **using** *tcHyp cHyp* **by** *simp*
 from *this* **have** $(\llbracket \eta \rrbracket_t (F t) \geq \llbracket \vartheta \rrbracket_t (F t))$ **by** *simp*
 also **have** $\llbracket \vartheta \preceq \eta \rrbracket_P c = (\llbracket \vartheta \rrbracket_t (F t) \leq \llbracket \eta \rrbracket_t (F t))$ **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 $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (vdiff \circ \pi_1) xfList) \otimes uInput) \upharpoonright_{\partial_P} \varphi \rrbracket_P st$
and *termVarsHyp:propVars* $\varphi \subseteq (UNIV - \text{varDiffs})$
and *listsHyp:map* $\pi_2 xfList = \text{map tval } uInput$
and *phi-p:P* $= \llbracket \varphi \rrbracket_P$
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 $P a \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem } xfList \text{ with } G) \longrightarrow P c)$
using *assms dInv-prelim* **by** *metis*
from this and aHyp have $\forall c. (a, c) \in (\text{ODEsystem } xfList \text{ with } G) \longrightarrow P c$ **by**
blast
thus $(a, b) \in \text{wp } (\text{ODEsystem } xfList \text{ with } G) \lceil P \rceil$
using *aHyp* **by** (*simp add: boxProgrPred-chrctrztn*)
qed

theorem dInvFinal:
assumes $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (vdiff \circ \pi_1) xfList) \otimes uInput) \upharpoonright_{\partial_P} \varphi \rrbracket_P st$
and *termVarsHyp:propVars* $\varphi \subseteq (UNIV - \text{varDiffs})$
and *listsHyp:map* $\pi_2 xfList = \text{map tval } uInput$
and *impls:* $\lceil P \rceil \subseteq \lceil F \rceil \wedge \lceil F \rceil \subseteq \lceil Q \rceil$
and *phi-f:F* $= \llbracket \varphi \rrbracket_P$
shows *PRE P (ODEsystem xfList with G) POST Q*
apply (*rule-tac* $C = \llbracket \varphi \rrbracket_P$ **in** *dCut*)
apply (*subgoal-tac* $\lceil F \rceil \subseteq \text{wp } (\text{ODEsystem } xfList \text{ with } G) \lceil F \rceil$, *simp*)
using *impls* **and** *phi-f* **apply** *blast*
apply (*subgoal-tac* *PRE F (ODEsystem xfList with G) POST F, simp*)
apply (*rule-tac* $\varphi = \varphi$ **and** $uInput = uInput$ **in** *dInv*)
prefer 5 **apply** (*subgoal-tac* *PRE P (ODEsystem xfList with G) 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

```

1.5 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:
   $PRE (\lambda s. s \text{ ''y''} < s \text{ ''x''} \wedge s \text{ ''v''} > 0)$ 
   $(ODEsystem [( \text{''x''}, (\lambda s. s \text{ ''v''}) )]) \text{ with } (\lambda s. True))$ 
   $POST (\lambda s. (s \text{ ''y''} < s \text{ ''x''}))$ 
apply(rule-tac uInput=[ $\lambda t s. s \text{ ''v''} \cdot t + s \text{ ''x''}$ ] 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= $\lambda x. s \text{ ''v''}$  and g'1= $\lambda x. 0$  in derivative-intros(190))
apply(rule-tac f'1= $\lambda x. 0$  and g'1= $\lambda x. 1$  in derivative-intros(193))
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: $\varphi_s \text{ solvesTheStoreIVP } [(x, \lambda s. s \ v), (v, \lambda s. s \ a)] \text{ withInitState } s$ 
and tHyp: $r \leq t$  and rHyp: $0 \leq r$  and distinct: $x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin \text{varDiffs}$ 
shows  $\varphi_s \ r \ v = s \ a \cdot r + s \ v$ 
proof—
from assms have 1:  $((\lambda t. \varphi_s \ t \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a)) \{0..t\} \text{ UNIV } \wedge \varphi_s \ 0 \ v = s \ v$ 
by (simp add: solvesStoreIVP-def)
from assms have obs: $\forall r \in \{0..t\}. \varphi_s \ r \ a = s \ a$ 
by(auto simp: solvesStoreIVP-def varDiffs-def)
have 2:  $((\lambda t. s \ a \cdot t + s \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a)) \{0..t\} \text{ UNIV}$ 
unfolding solves-ode-def apply(subgoal-tac  $((\lambda x. s \ a \cdot x + s \ v) \text{ has-vderiv-on } (\lambda x. s \ a)) \{0..t\})$ 
using obs apply (simp add: has-vderiv-on-def) by(rule galilean-transform)
have 3: unique-on-bounded-closed 0  $\{0..t\} (s \ v) (\lambda t \ r. \varphi_s \ t \ a) \text{ UNIV (if } t = 0 \text{ then } 1 \text{ else } 1/(t+1))$ 
apply(simp add: ubc-definitions del: comp-apply, rule conjI)
using rHyp tHyp obs apply(simp-all del: comp-apply)
apply(clarify, rule continuous-intros) prefer 3 apply safe

```

```

apply(rule continuous-intros)
apply(auto intro: continuous-intros)
by (metis continuous-on-const continuous-on-eq)
thus  $\varphi_s \ r \ v = s \ a \cdot r + s \ v$ 
apply(rule-tac unique-on-bounded-closed.unique-solution[of 0 {0..t} s v
  ( $\lambda t \ r. \ \varphi_s \ t \ a$ ) UNIV (if  $t = 0$  then 1 else  $1 / (t + 1)$ ) ( $\lambda t. \ \varphi_s \ t \ v$ )])
using rHyp tHyp 1 2 and 3 by auto
qed

lemma motion-with-constant-acceleration:
  PRE ( $\lambda s. \ s \ "y" < s \ "x" \wedge s \ "v" \geq 0 \wedge s \ "a" > 0$ )
  (ODEsystem [( $"x", (\lambda s. \ s \ "v")$ ), ( $"v", (\lambda s. \ s \ "a")$ )] with ( $\lambda s. \ \text{True}$ ))
  POST ( $\lambda s. \ (s \ "y" < s \ "x')$ )
apply(rule-tac uInput=[ $\lambda t \ s. \ s \ "a" \cdot t^2/2 + s \ "v" \cdot t + s \ "x",$ 
   $\lambda t \ s. \ s \ "a" \cdot t + s \ "v"$ ] in dSolve-toSolveUBC)
prefer 9 subgoal by(simp add: wp-trafo vdiff-def add-strict-increasing2)
prefer 6 subgoal
  apply(simp add: vdiff-def, clarify, rule conjI)
  by(rule galilean-transform)+
prefer 6 subgoal
  apply(simp add: vdiff-def, safe)
  by(rule continuous-intros)+
prefer 6 subgoal
  apply(simp add: vdiff-def, safe)
  subgoal for  $s \ \varphi_s \ t \ r$  apply(rule flow-vel-is-galilean-vel[of  $\varphi_s \ "x" \dots t$ ])
  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: $\varphi_s$  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: $x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin$ 
 $\text{varDiffs}$ 
shows  $\varphi_s \ r \ v = s \ v - s \ a \cdot r$ 
proof –
from assms have 1:(( $\lambda t. \ \varphi_s \ t \ v$ ) solves-ode ( $\lambda t \ r. \ - \ \varphi_s \ t \ a$ )) {0..t} UNIV  $\wedge \varphi_s$ 
 $0 \ v = s \ v$ 
by (simp add: solvesStoreIVP-def)
from assms have obs: $\forall \ r \in \{0..t\}. \ \varphi_s \ r \ a = s \ a$ 
by(auto simp: solvesStoreIVP-def varDiffs-def)
have 2:(( $\lambda t. \ - \ s \ a \cdot t + s \ v$ ) solves-ode ( $\lambda t \ r. \ - \ \varphi_s \ t \ a$ )) {0..t} UNIV
unfolding solves-ode-def apply(subgoal-tac (( $\lambda x. \ - \ s \ a \cdot x + s \ v$ ) has-vderiv-on
( $\lambda x. \ - \ s \ a$ )) {0..t}))
using obs apply (simp add: has-vderiv-on-def) by(rule galilean-transform)
have 3:unique-on-bounded-closed 0 {0..t} ( $s \ v$ ) ( $\lambda t \ r. \ - \ \varphi_s \ t \ a$ ) UNIV (if  $t = 0$ 
then 1 else  $1/(t+1)$ )

```

```

apply(simp add: ubc-definitions del: comp-apply, rule conjI)
using rHyp tHyp obs apply(simp-all del: comp-apply)
apply(clarify, rule continuous-intros) prefer 3 apply safe
apply(rule continuous-intros)+
apply(auto intro: continuous-intros)
by (metis continuous-on-const continuous-on-eq)
thus  $\varphi_s \ r \ v = s \ v - s \ a \cdot r$ 
apply(rule-tac unique-on-bounded-closed.unique-solution[of 0 {0..t} s v
  ( $\lambda t \ r. - \varphi_s \ t \ a$ ) UNIV (if  $t = 0$  then 1 else 1 / (t + 1)) ( $\lambda t. \varphi_s \ t \ v$ )])
using rHyp tHyp 1 2 and 3 by auto
qed

lemma single-hop-ball:
  PRE ( $\lambda s. 0 \leq s \ "x" \wedge s \ "x" = H \wedge s \ "v" = 0 \wedge s \ "g" > 0 \wedge 1 \geq c \wedge c \geq 0$ )
  (((ODEsystem [( $"x", \lambda s. s \ "v"$ ), ( $"v", \lambda s. - s \ "g"$ )] with ( $\lambda s. 0 \leq s \ "x"$ )));
  (IF ( $\lambda s. s \ "x" = 0$ ) THEN ( $"v" ::= (\lambda s. - c \cdot s \ "v")$ ) ELSE ( $"v" ::= (\lambda s. s \ "v")$ ) FI))
  POST ( $\lambda s. 0 \leq s \ "x" \wedge s \ "x" \leq H$ )
  apply(simp, subst dS[of [ $\lambda t \ s. - s \ "g" \cdot t^2/2 + s \ "v" \cdot t + s \ "x", \lambda t \ s. - s \ "g" \cdot t + s \ "v"$ ]])
  — Given solution is actually a solution.
  apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton,
    safe)
  apply(rule galilean-transform-eq, simp)+
  apply(rule galilean-transform)+
  — Uniqueness of the flow.
  apply(rule ubcStoreUniqueSol, simp)
  apply(simp add: vdiff-def del: comp-apply)
  apply(auto intro: continuous-intros del: comp-apply)[1]
  apply(rule continuous-intros)+
  apply(simp add: vdiff-def, safe)
  apply(clarsimp) subgoal for  $s \ X \ t \ \tau$ 
  apply(rule flow-vel-is-galilean-vel2[of X  $"x"$ ])
  by(simp-all add: varDiffs-def vdiff-def)
  apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def)
  apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def
    has-vderiv-on-singleton galilean-transform-eq galilean-transform)
  — Relation Between the guard and the postcondition.
  by(auto simp: vdiff-def p2r-def)

— Example of hybrid program verified with differential weakening.
lemma system-where-the-guard-implies-the-postcondition:
  PRE ( $\lambda s. s \ "x" = 0$ )
  (ODEsystem [( $"x", (\lambda s. s \ "x" + 1)$ )] with ( $\lambda s. s \ "x" \geq 0$ ))
  POST ( $\lambda s. s \ "x" \geq 0$ )
using dWeakening by blast

```

lemma system-where-the-guard-implies-the-postcondition2:

```

    PRE ( $\lambda s. s \text{''}x'' = 0$ )
    (ODEsystem [( $\text{''}x''$ , ( $\lambda s. s \text{''}x'' + 1$ ))]) with ( $\lambda s. s \text{''}x'' \geq 0$ )
    POST ( $\lambda s. s \text{''}x'' \geq 0$ )
  apply (clarify, simp add: p2r-def)
  apply (simp add: rel-ad-def rel-antidomain-kleene-algebra.addual.ars-r-def)
  apply (simp add: rel-antidomain-kleene-algebra.fbox-def)
  apply (simp add: relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def)
  by auto

```

— Example of system proved with a differential invariant.

lemma *circular-motion*:

```

    PRE ( $\lambda s. (s \text{''}x'' \cdot (s \text{''}x'') + (s \text{''}y'' \cdot (s \text{''}y'') - (s \text{''}r'' \cdot (s \text{''}r'')) = 0$ )
    (ODEsystem [( $\text{''}x''$ , ( $\lambda s. s \text{''}y''$ )), ( $\text{''}y''$ , ( $\lambda s. -s \text{''}x''$ ))]) with G
    POST ( $\lambda s. (s \text{''}x'' \cdot (s \text{''}x'') + (s \text{''}y'' \cdot (s \text{''}y'') - (s \text{''}r'' \cdot (s \text{''}r'')) = 0$ )
  apply (rule-tac  $\eta = (t_V \text{''}x'') \odot (t_V \text{''}x'') \oplus (t_V \text{''}y'') \odot (t_V \text{''}y'') \oplus (\ominus (t_V \text{''}r'')) \odot (t_V \text{''}r'')$ )
  and uInput=[ $t_V \text{''}y''$ ,  $\ominus (t_V \text{''}x'')$ ] in dInvForTrms)
  apply (simp-all add: vdiff-def varDiffs-def)
  apply (clarsimp, erule-tac  $x = \text{''}r''$  in allE)
  by simp

```

— Example of systems proved with differential invariants, cuts and weakenings.

declare *d-p2r* [simp del]

lemma *motion-with-constant-velocity-and-invariants*:

```

    PRE ( $\lambda s. s \text{''}x'' > s \text{''}y'' \wedge s \text{''}v'' > 0$ )
    (ODEsystem [( $\text{''}x''$ , ( $\lambda s. s \text{''}v''$ ))]) with ( $\lambda s. \text{True}$ )
    POST ( $\lambda s. s \text{''}x'' > s \text{''}y''$ )
  apply (rule-tac  $C = \lambda s. s \text{''}v'' > 0$  in dCut)
  apply (rule-tac  $\varphi = (t_C 0) \prec (t_V \text{''}v'')$  and uInput=[ $t_V \text{''}v''$ ] in dInvFinal)
  apply (simp-all add: vdiff-def varDiffs-def, clarify, erule-tac  $x = \text{''}v''$  in allE, simp)
  apply (rule-tac  $C = \lambda s. s \text{''}x'' > s \text{''}y''$  in dCut)
  apply (rule-tac  $\varphi = (t_V \text{''}y'') \prec (t_V \text{''}x'')$  and uInput=[ $t_V \text{''}v''$ ] and
     $F = \lambda s. s \text{''}x'' > s \text{''}y''$  in dInvFinal)
  apply (simp-all add: vdiff-def varDiffs-def, clarify, erule-tac  $x = \text{''}y''$  in allE, simp)
  using dWeakening by simp

```

lemma *motion-with-constant-acceleration-and-invariants*:

```

    PRE ( $\lambda s. s \text{''}y'' < s \text{''}x'' \wedge s \text{''}v'' \geq 0 \wedge s \text{''}a'' > 0$ )
    (ODEsystem [( $\text{''}x''$ , ( $\lambda s. s \text{''}v''$ )), ( $\text{''}v''$ , ( $\lambda s. s \text{''}a''$ ))]) with ( $\lambda s. \text{True}$ )
    POST ( $\lambda s. (s \text{''}y'' < s \text{''}x'')$ )
  apply (rule-tac  $C = \lambda s. s \text{''}a'' > 0$  in dCut)
  apply (rule-tac  $\varphi = (t_C 0) \prec (t_V \text{''}a'')$  and uInput=[ $t_V \text{''}v''$ ,  $t_V \text{''}a''$ ] in dInvFinal)
  apply (simp-all add: vdiff-def varDiffs-def, clarify, erule-tac  $x = \text{''}a''$  in allE, simp)
  apply (rule-tac  $C = \lambda s. s \text{''}v'' \geq 0$  in dCut)
  apply (rule-tac  $\varphi = (t_C 0) \preceq (t_V \text{''}v'')$  and uInput=[ $t_V \text{''}v''$ ,  $t_V \text{''}a''$ ] in dInvFi-
    nal)
  apply (simp-all add: vdiff-def varDiffs-def)
  apply (rule-tac  $C = \lambda s. s \text{''}x'' > s \text{''}y''$  in dCut)
  apply (rule-tac  $\varphi = (t_V \text{''}y'') \prec (t_V \text{''}x'')$  and uInput=[ $t_V \text{''}v''$ ,  $t_V \text{''}a''$ ] in dInv-

```

Final)
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 ($\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' = H \wedge s \text{ ''}v'' = 0 \wedge s \text{ ''}g'' > 0 \wedge 1 \geq c \wedge c \geq 0$)
 $((ODEsystem [(x'', \lambda s. s \text{ ''}v''), (v'', \lambda s. - s \text{ ''}g'')] \text{ with } (\lambda s. 0 \leq s \text{ ''}x'')));$
 $(IF (\lambda s. s \text{ ''}x'' = 0) THEN (v'' ::= (\lambda s. - c \cdot s \text{ ''}v'')) ELSE (v'' ::= (\lambda s. s \text{ ''}v'')) FI))$
POST ($\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H$)
apply(*simp add: d-p2r, subgoal-tac rdom* [$\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' = H \wedge s \text{ ''}v'' = 0 \wedge 0 < s \text{ ''}g'' \wedge c \leq 1 \wedge 0 \leq c$]
 $\subseteq wp (ODEsystem [(x'', \lambda s. s \text{ ''}v''), (v'', \lambda s. - s \text{ ''}g'')] \text{ with } (\lambda s. 0 \leq s \text{ ''}x''))$
 $(inf (sup (- (\lambda s. s \text{ ''}x'' = 0)) (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H)) (sup (\lambda s. s \text{ ''}x'' = 0) (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H)))$)
apply(*simp add: d-p2r, rule-tac* $C = \lambda s. s \text{ ''}g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C 0) \prec (t_V \text{ ''}g'')$ **and** *uInput* $= [t_V \text{ ''}v'', \ominus t_V \text{ ''}g'']$ **in** *dInvFinal*)
apply(*simp-all add: vdiff-def varDiffs-def, clarify, erule-tac* $x = "g"$ **in** *allE, simp*)
apply(*rule-tac* $C = \lambda s. s \text{ ''}v'' \leq 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V \text{ ''}v'') \preceq (t_C 0)$ **and** *uInput* $= [t_V \text{ ''}v'', \ominus t_V \text{ ''}g'']$ **in** *dInvFinal*)
apply(*simp-all add: vdiff-def varDiffs-def*)
apply(*rule-tac* $C = \lambda s. s \text{ ''}x'' \leq H$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V \text{ ''}x'') \preceq (t_C H)$ **and** *uInput* $= [t_V \text{ ''}v'', \ominus t_V \text{ ''}g'']$ **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*: $0 \leq x \implies 0 < g \implies 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \implies (x :: real) \leq H$

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 ( $\lambda s. 0 \leq s \cdot x'' \wedge s \cdot x'' = H \wedge s \cdot v'' = 0 \wedge s \cdot g'' > 0$ )
((ODEsystem [( $x'', \lambda s. s \cdot v''$ ), ( $v'', \lambda s. -s \cdot g''$ )] with ( $\lambda s. 0 \leq s \cdot x''$ )));
(IF ( $\lambda s. s \cdot x'' = 0$ ) THEN ( $v'' ::= (\lambda s. -s \cdot v'')$ ) ELSE (Id FI))*
POST ( $\lambda s. 0 \leq s \cdot x'' \wedge s \cdot x'' \leq H$ )
apply(rule rel-antidomain-kleene-algebra.fbox-starI[of - [ $\lambda s. 0 \leq s \cdot x'' \wedge 0 < s \cdot g'' \wedge$ 
 $2 \cdot s \cdot g'' \cdot s \cdot x'' = 2 \cdot s \cdot g'' \cdot H - (s \cdot v'' \cdot s \cdot v'')$ ]])
apply(simp, simp add: d-p2r)
apply(subgoal-tac
  rdom [ $\lambda s. 0 \leq s \cdot x'' \wedge 0 < s \cdot g'' \wedge 2 \cdot s \cdot g'' \cdot s \cdot x'' = 2 \cdot s \cdot g'' \cdot H - s \cdot v'' \cdot s \cdot v''$ ]
   $\subseteq wp$  (ODEsystem [( $x'', \lambda s. s \cdot v''$ ), ( $v'', \lambda s. -s \cdot g''$ )] with ( $\lambda s. 0 \leq s \cdot x''$ ))
)
[inf (sup (- ( $\lambda s. s \cdot x'' = 0$ )) ( $\lambda s. 0 \leq s \cdot x'' \wedge 0 < s \cdot g'' \wedge 2 \cdot s \cdot g'' \cdot s \cdot x''$ 
=
 $2 \cdot s \cdot g'' \cdot H - s \cdot v'' \cdot s \cdot v''$ ))
(sup ( $\lambda s. s \cdot x'' = 0$ ) ( $\lambda s. 0 \leq s \cdot x'' \wedge 0 < s \cdot g'' \wedge 2 \cdot s \cdot g'' \cdot s \cdot x'' =$ 
 $2 \cdot s \cdot g'' \cdot H - s \cdot v'' \cdot s \cdot v''$ ))]
apply(simp add: d-p2r)
apply(rule-tac  $C = \lambda s. s \cdot g'' > 0$  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  $C = \lambda s. 2 \cdot s \cdot g'' \cdot s \cdot x'' = 2 \cdot s \cdot g'' \cdot H - s \cdot v'' \cdot s \cdot v''$  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

theory *flow-locales*

imports *Ordinary-Differential-Equations.Initial-Value-Problem*

begin

2 Flow Locales

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

```

declare unique-on-bounded-closed-def [ubc-definitions]
and unique-on-bounded-closed-axioms-def [ubc-definitions]
and unique-on-closed-def [ubc-definitions]
and compact-interval-def [ubc-definitions]

```



```

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

lemma(in unique-on-bounded-closed) unique-on-bounded-closed-on-compact-subset:
  assumes  $t0 \in T'$  and  $x0 \in X$  and  $T' \subseteq T$  and compact-interval  $T'$ 
  shows unique-on-bounded-closed  $t0$   $T'$   $x0$   $f$   $X$   $L$ 
  apply(unfold-locales)
  using  $\langle \text{compact-interval } T' \rangle$  unfolding ubc-definitions apply simp+
  using  $\langle t0 \in T' \rangle$  apply simp
  using  $\langle x0 \in X \rangle$  apply simp
  using  $\langle T' \subseteq T \rangle$  self-mapping apply blast
  using  $\langle T' \subseteq T \rangle$  continuous apply(meson Sigma-mono continuous-on-subset subsetI)
  using  $\langle T' \subseteq T \rangle$  lipschitz apply blast
  using  $\langle T' \subseteq T \rangle$  lipschitz-bound by blast

```

The first locale imposes conditions for applying the Picard-Lindelöf theorem following the people who created the Ordinary Differential Equations entry in the AFP.

```

locale picard-ivp = continuous-rhs  $T$   $S$   $f$  + global-lipschitz  $T$   $S$   $f$   $L$ 
  for  $f::\text{real} \Rightarrow ('a::\text{banach}) \Rightarrow 'a$  and  $T::\text{real set}$  and  $S$   $L$  +
  fixes  $t0::\text{real}$ 
  assumes init-time:  $t0 \in T$ 
    and nonempty-time:  $T \neq \{\}$ 
    and interval-time: is-interval  $T$ 
    and compact-time: compact  $T$ 
    and lipschitz-bound:  $\bigwedge s t. s \in T \implies t \in T \implies \text{abs } (s - t) * L < 1$ 
    and closed-domain: closed  $S$ 
    and solution-in-domain:  $\bigwedge x s t. t \in T \implies x \ t0 = s \implies x \in \{t0 \dots t\} \rightarrow S$ 
   $\implies$ 
    continuous-on  $\{t0 \dots t\}$   $x \implies x \ t0 + \text{ivl-integral } t0 \ t (\lambda t. f \ t \ (x \ t)) \in S$ 
begin

```

```

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

```

```

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

```

```

lemma subinterval:
  assumes  $t \in T$ 
  obtains  $t1$  where  $\{t .. t1\} \subseteq T$ 

```

```

using assms interval-subset-is-interval interval-time by fastforce

lemma subsegment:
  assumes  $t1 \in T$  and  $t2 \in T$ 
  shows  $\{t1 \dashv\vdash t2\} \subseteq T$ 
  using assms closed-segment-subset-domain by blast

lemma is-ubc:
  assumes  $s \in S$ 
  shows unique-on-bounded-closed  $t0\ T\ s\ f\ S\ L$ 
  using assms unfolding ubc-definitions apply safe
  prefer 6 using solution-in-domain apply simp
  prefer 2 using nonempty-time apply fastforce
  by(auto simp: compact-time interval-time init-time
      closed-domain lipschitz lipschitz-bound continuous)

lemma unique-solution:
  assumes ( $x$  solves-ode  $f$ )  $T\ S$  and  $x\ t0 = s$ 
  and ( $y$  solves-ode  $f$ )  $T\ S$  and  $y\ t0 = s$ 
  and  $s \in S$  and  $t \in T$ 
  shows  $x\ t = y\ t$ 
  using unique-on-bounded-closed.unique-solution is-ubc assms by blast

abbreviation  $\phi\ t\ s \equiv$  (apply-bcontfun (unique-on-bounded-closed.fixed-point  $t0\ T\ s\ f\ S$ ))  $t$ 

lemma fixed-point-solves:
  assumes  $s \in S$ 
  shows ( $\lambda\ t.\ \phi\ t\ s$ ) solves-ode  $f$ )  $T\ S$  and  $\phi\ t0\ s = s$ 
  using assms is-ubc unique-on-bounded-closed.fixed-point-solution apply(metis
  (full-types))
  using assms is-ubc unique-on-bounded-closed.fixed-point-iv by(metis (full-types))

lemma fixed-point-usolves:
  assumes ( $x$  solves-ode  $f$ )  $T\ S$  and  $x\ t0 = s$  and  $t \in T$ 
  shows  $x\ t = \phi\ t\ s$ 
  using assms(1,2) unfolding solves-ode-def apply(subgoal-tac  $s \in S$ )
  using unique-solution fixed-point-solves assms apply blast
  unfolding Pi-def using init-time by auto

end

The next locale particularizes the previous one to an initial time equal to 0. Thus making the function that maps every initial point to its solution a (local) “flow”.

locale local-flow = picard-ivp ( $\lambda\ t.\ f$ )  $T\ S\ L\ 0$  for  $f :: ('a :: \textit{banach}) \Rightarrow 'a$  and  $S\ T\ L +$ 
  fixes  $\varphi :: \textit{real} \Rightarrow 'a \Rightarrow 'a$ 

```

assumes *ivp*: $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } (\lambda t. f)) T \ S \wedge \varphi \ 0 \ s = s$
begin

lemma *is-fixed-point*:
assumes $s \in S$ **and** $t \in T$
shows $\varphi \ t \ s = \text{phi } t \ s$
apply(*rule fixed-point-usolves*)
using *ivp assms init-time* **by** *simp-all*

theorem *solves*:
assumes $s \in S$
shows $((\lambda t. \varphi \ t \ s) \text{ solves-ode } (\lambda t. f)) T \ S$
using *assms init-time fixed-point-solves(1)* **and** *is-fixed-point* **by** *auto*

theorem *on-init-time*:
assumes $s \in S$
shows $\varphi \ 0 \ s = s$
using *assms init-time fixed-point-solves(2)* **and** *is-fixed-point* **by** *auto*

lemma *is-banach-endo*:
assumes $s \in S$ **and** $t \in T$
shows $\varphi \ t \ s \in S$
apply(*rule-tac A=T in Pi-mem*)
using *assms solves*
unfolding *solves-ode-def* **by** *auto*

lemma *solves-on-compact-subset*:
assumes $T' \subseteq T$ **and** *compact-interval* T' **and** $0 \in T'$
shows $t \in T' \implies (x \text{ solves-ode } (\lambda t. f)) T' \ S \implies \varphi \ t \ (x \ 0) = x \ t$
proof–
fix t **and** x **assume** $t \in T'$ **and** $x \text{ solves } (x \text{ solves-ode } (\lambda t. f)) T' \ S$
from *this* **and** $\langle 0 \in T' \rangle$ **have** $x \ 0 \in S$ **unfolding** *solves-ode-def* **by** *blast*
then **have** $((\lambda \tau. \varphi \ \tau \ (x \ 0)) \text{ solves-ode } (\lambda \tau. f)) T \ S$ **using** *solves* **by** *blast*
hence *flow-solves*: $((\lambda \tau. \varphi \ \tau \ (x \ 0)) \text{ solves-ode } (\lambda \tau. f)) T' \ S$
using $\langle T' \subseteq T \rangle$ *solves-ode-on-subset* **by** (*metis subset-eq*)
have *unique-on-bounded-closed* $0 \ T \ (x \ 0) \ (\lambda \tau. f) \ S \ L$
using *is-ubc* **and** $\langle x \ 0 \in S \rangle$ **by** *blast*
then **have** *unique-on-bounded-closed* $0 \ T' \ (x \ 0) \ (\lambda \tau. f) \ S \ L$
using *unique-on-bounded-closed.unique-on-bounded-closed-on-compact-subset*
 $\langle 0 \in T' \rangle \langle x \ 0 \in S \rangle \langle T' \subseteq T \rangle$ **and** $\langle \text{compact-interval } T' \rangle$ **by** *blast*
moreover **have** $\varphi \ 0 \ (x \ 0) = x \ 0$
using *on-init-time* **and** $\langle x \ 0 \in S \rangle$ **by** *blast*
ultimately **show** $\varphi \ t \ (x \ 0) = x \ t$
using *unique-on-bounded-closed.unique-solution flow-solves x-solves* **and** $\langle t \in T' \rangle$ **by** *blast*
qed
end

lemma *flow-on-compact-subset*:
assumes *flow-from-Y:local-flow f S T' L φ and $T \subseteq T'$ and compact-interval T and $0 \in T$*
shows *local-flow f S T L φ*
using *assms unfolding local-flow-def picard-ivp-def ubc-definitions* **apply**(*unfold-locales, safe*)
apply(*meson Sigma-mono continuous-on-subset subsetI*) **apply** *blast*
unfolding *picard-ivp-axioms-def* **apply**(*simp add: subset-eq, blast*)
unfolding *local-flow-axioms-def* **using** *solves-ode-on-subset* **by** *fastforce*

The last locale shows that the function introduced in its predecessor is indeed a flow. That is, it is a group action on the additive part of the real numbers.

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

lemma *add-flow-solves:(($\lambda \tau. \varphi (\tau + t) s$) solves-ode ($\lambda t. f$)) UNIV UNIV*
unfolding *solves-ode-def* **apply** *safe*
apply(*subgoal-tac (($\lambda \tau. \varphi \tau s$) \circ ($\lambda \tau. \tau + t$) has-vderiv-on*
*($\lambda x. (\lambda \tau. 1) x *_R (\lambda t. f (\varphi t s)) ((\lambda \tau. \tau + t) x)))$ UNIV, simp add: comp-def)*
apply(*rule has-vderiv-on-compose*)
using *solves min-max-interval* **unfolding** *solves-ode-def* **apply** *auto[1]*
apply(*rule-tac f'1= $\lambda x. 1$ and g'1= $\lambda x. 0$ in derivative-intros(190)*)
apply(*rule derivative-intros, simp*)
by *auto*

theorem *flow-is-group-action*:

shows $\varphi 0 s = s$
and $\varphi (t1 + t2) s = \varphi t1 (\varphi t2 s)$
proof–
show $\varphi 0 s = s$ **using** *on-init-time* **by** *simp*
have $g1:\varphi (0 + t2) s = \varphi t2 s$ **by** *simp*
have $g2:((\lambda \tau. \varphi (\tau + t2) s) \text{ solves-ode } (\lambda t. f)) \text{ UNIV UNIV}$
using *add-flow-solves* **by** *simp*
have $h0:\varphi t2 s \in \text{UNIV}$
using *is-banach-endo* **by** *simp*
hence $h1:\varphi 0 (\varphi t2 s) = \varphi t2 s$
using *on-init-time* **by** *simp*
have $h2:((\lambda \tau. \varphi \tau (\varphi t2 s)) \text{ solves-ode } (\lambda t. f)) \text{ UNIV UNIV}$
apply(*rule-tac S=UNIV and Y=UNIV in solves-ode-on-subset*)
using $h0$ **solves** **by** *auto*
from $g1 g2 h1$ **and** $h2$ **have** $\bigwedge t. \varphi (t + t2) s = \varphi t (\varphi t2 s)$
using *unique-on-bounded-closed.unique-solution is-ubc* **by** *blast*
thus $\varphi (t1 + t2) s = \varphi t1 (\varphi t2 s)$ **by** *simp*
qed

end

lemma *localize-global-flow*:

assumes *global-flow f L φ and compact-interval T and closed S*

```

shows local-flow f S T L  $\varphi$ 
using assms unfolding global-flow-def local-flow-def
picard-ivp-def picard-ivp-axioms-def by simp

```

2.1 Example

Finally, we exemplify a procedure for introducing pairs of vector fields and their respective flows using the previous locales.

```

lemma constant-is-picard-ivp:  $0 \leq t \implies \text{picard-ivp } (\lambda t s. c) \{0..t\} \text{ UNIV } (1 / (t + 1)) 0$ 
  unfolding picard-ivp-def picard-ivp-axioms-def ubc-definitions
  by (simp add: nonempty-set-def lipschitz-on-def, clarsimp, simp)

```

```

lemma line-solves-constant:  $((\lambda \tau. x + \tau *_{\mathbb{R}} c) \text{ solves-ode } (\lambda t s. c)) \{0..t\} \text{ UNIV}$ 
  unfolding solves-ode-def apply simp
  apply (rule-tac f'1= $\lambda x. 0$  and g'1= $\lambda x. c$  in derivative-intros(190))
  apply (rule derivative-intros, simp)+
  by simp-all

```

```

lemma line-is-local-flow:
 $0 \leq t \implies \text{local-flow } (\lambda s. (c::'a::\text{banach})) \text{ UNIV } \{0..t\} (1/(t + 1)) (\lambda t x. x + t *_{\mathbb{R}} c)$ 
  unfolding local-flow-def local-flow-axioms-def apply safe
  using constant-is-picard-ivp apply blast
  using line-solves-constant by auto

```

```

end
theory cat2rel-pre
  imports
    ../flow-locales
    ../../afpModified/VC-KAD

```

```
begin
```

3 Preliminaries

This file presents a miscellaneous collection of preliminary lemmas for verification of Hybrid Systems in Isabelle.

— 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 ( $r$ )

```

3.1 Weakest Liberal Preconditions

```

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

```

definition $f2r :: ('a \Rightarrow 'b \text{ set}) \Rightarrow ('a \times 'b) \text{ set } (\mathcal{R})$ **where**

$\mathcal{R} f = \{(x,y). y \in f x\}$

lemma $wp\text{-}rel:wp R \lceil P \rceil = \lceil \lambda x. \forall y. (x,y) \in R \longrightarrow P y \rceil$

proof–

have $\lfloor wp R \lceil P \rceil \rfloor = \lfloor \lceil \lambda x. \forall y. (x,y) \in R \longrightarrow P y \rceil \rfloor$

by (*simp add: wp-trafo pointfree-idE*)

thus $wp R \lceil P \rceil = \lceil \lambda x. \forall y. (x,y) \in R \longrightarrow P y \rceil$

by (*metis (no-types, lifting) wp-simp d-p2r pointfree-idE prp*)

qed

corollary $wp\text{-}relD:(x,x) \in wp R \lceil P \rceil \Longrightarrow \forall y. (x,y) \in R \longrightarrow P y$

proof–

assume $(x,x) \in wp R \lceil P \rceil$

hence $(x,x) \in \lceil \lambda x. \forall y. (x,y) \in R \longrightarrow P y \rceil$ **using** *wp-rel* **by** *auto*

thus $\forall y. (x,y) \in R \longrightarrow P y$ **by** (*simp add: p2r-def*)

qed

lemma $p2r\text{-}r2p\text{-}wp\text{-}sym:wp R P = \lceil \lfloor wp R P \rfloor \rceil$

using *d-p2r wp-simp* **by** *blast*

lemma $p2r\text{-}r2p\text{-}wp:\lceil \lfloor wp R P \rfloor \rceil = wp R P$

by(*rule sym, subst p2r-r2p-wp-sym, simp*)

abbreviation $vec\text{-}upd :: ('a \wedge 'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a \wedge 'b \text{ } (- (2[-] ::= -))$ [70, 65] 61)

where

$x[i ::= a] \equiv (\chi j. (if j = i then a else (x \$ j)))$

abbreviation $assign :: 'b \Rightarrow ('a \wedge 'b \Rightarrow 'a) \Rightarrow ('a \wedge 'b) \text{ rel } ((2[-] ::= -))$ [70, 65] 61) **where**

$[x ::= expr] \equiv \{(s, s[x ::= expr s]) \mid s. True\}$

lemma $wp\text{-}assign [simp]: wp ([x ::= expr]) \lceil Q \rceil = \lceil \lambda s. Q (s[x ::= expr s]) \rceil$

by(*auto simp: rel-antidomain-kleene-algebra.fbox-def rel-ad-def p2r-def*)

lemma $wp\text{-}assign\text{-}var [simp]: \lfloor wp ([x ::= expr]) \lceil Q \rceil \rfloor = (\lambda s. Q (s[x ::= expr s]))$

by(*subst wp-assign, simp add: pointfree-idE*)

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 $d p \leq |x^*| q$

proof–

from $\langle d i \leq |x| i \rangle$ **have** $d i \leq |x| (d i)$

using *local.fbox-simp* **by** *auto*

hence $|1| p \leq |x^*| i$ **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

lemma *rel-ad-mka-starI*:

assumes $P \subseteq I$ **and** $I \subseteq wp\ R\ I$ **and** $I \subseteq Q$

shows $P \subseteq wp\ (R^*)\ Q$

proof –

have $wp\ R\ I \subseteq Id$

by (*simp add: rel-antidomain-kleene-algebra.a-subid rel-antidomain-kleene-algebra.fbox-def*)

hence $P \subseteq Id$ **using** *assms(1,2)* **by** *blast*

from *this* **have** $rdom\ P = P$ **by** (*metis d-p2r p2r-surj*)

also have $rdom\ P \subseteq wp\ (R^*)\ Q$

by (*metis (wp\ R\ I \subseteq Id) assms d-p2r p2r-surj*)

rel-antidomain-kleene-algebra.dka.dom-iso rel-antidomain-kleene-algebra.fbox-starI)

ultimately show *?thesis* **by** *blast*

qed

3.2 Real Numbers and Derivatives

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

by *auto*

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

by *auto*

lemma *sqrt-le-itself*: $1 \leq x \implies \text{sqrt}\ x \leq x$

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* $(\text{real } n) \leq \text{real } n$

by (*metis (full-types) abs-of-nat le-square of-nat-mono of-nat-mult real-sqrt-abs2 real-sqrt-le-iff*)

lemma *semiring-factor-left*: $a \cdot b + a \cdot c = a \cdot ((b :: ('a :: \text{semiring})) + c)$

by (*subst Groups.algebra-simps(18), simp*)

lemma *sin-cos-squared-add3*: $(x :: ('a :: \{\text{banach}, \text{real-normed-field}\})) \cdot (\sin\ t)^2 + x \cdot (\cos\ t)^2 = x$

by (*subst semiring-factor-left, subst sin-cos-squared-add, simp*)

lemma *sin-cos-squared-add4*: $(x :: ('a :: \{\text{banach}, \text{real-normed-field}\})) \cdot (\cos\ t)^2 + x \cdot (\sin\ t)^2 = x$

by (*subst semiring-factor-left, subst sin-cos-squared-add2, simp*)

lemma *[simp]*: $((x :: \text{real}) \cdot \cos\ t - y \cdot \sin\ t)^2 + (x \cdot \sin\ t + y \cdot \cos\ t)^2 = x^2 + y^2$

proof –

have $(x \cdot \cos\ t - y \cdot \sin\ t)^2 = x^2 \cdot (\cos\ t)^2 + y^2 \cdot (\sin\ t)^2 - 2 \cdot (x \cdot \cos\ t) \cdot (y \cdot \sin\ t)$

by(simp add: power2-diff power-mult-distrib)
also have $(x \cdot \sin t + y \cdot \cos t)^2 = y^2 \cdot (\cos t)^2 + x^2 \cdot (\sin t)^2 + 2 \cdot (x \cdot \cos t) \cdot (y \cdot \sin t)$
by(simp add: power2-sum power-mult-distrib)
ultimately show $(x \cdot \cos t - y \cdot \sin t)^2 + (x \cdot \sin t + y \cdot \cos t)^2 = x^2 + y^2$
by (simp add: Groups.mult-ac(2) Groups.mult-ac(3) right-diff-distrib sin-squared-eq)

qed

lemma closed-segment-mvt:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $(\bigwedge r. r \in \{a \dots b\} \implies (f \text{ has-derivative } f' r) \text{ (at } r \text{ within } \{a \dots b\}))$ **and**
 $a \leq b$
shows $\exists r \in \{a \dots b\}. f b - f a = f' r (b - a)$
using *assms closed-segment-eq-real-ivl* **and** *mvt-very-simple* **by** *auto*

lemma convergences-solves-vec-nth:

assumes $((\lambda y. (\varphi y - \varphi (\text{netlimit (at } x \text{ within } \{0..t\}))) - (y - \text{netlimit (at } x \text{ within } \{0..t\}))) *_{\mathbb{R}} f (\varphi x)) /_{\mathbb{R}}$
 $|y - \text{netlimit (at } x \text{ within } \{0..t\})| \longrightarrow 0$ $(\text{at } x \text{ within } \{0..t\})$ **(is** $((\lambda y. ?f y) \longrightarrow 0) ?_{\text{net}}$
shows $((\lambda y. (\varphi y \$ i - \varphi (\text{netlimit (at } x \text{ within } \{0..t\}))) \$ i - (y - \text{netlimit (at } x \text{ within } \{0..t\}))) *_{\mathbb{R}} f (\varphi x) \$ i) /_{\mathbb{R}}$
 $|y - \text{netlimit (at } x \text{ within } \{0..t\})| \longrightarrow 0$ $(\text{at } x \text{ within } \{0..t\})$ **(is** $((\lambda y. ?g y i) \longrightarrow 0) ?_{\text{net}}$

proof–

from *assms* **have** $((\lambda y. ?f y \$ i) \longrightarrow 0 \$ i) ?_{\text{net}}$ **by** (rule *tendsto-vec-nth*)
also have $(\lambda y. ?f y \$ i) = (\lambda y. ?g y i)$ **by** *auto*
ultimately show $((\lambda y. ?g y i) \longrightarrow 0) ?_{\text{net}}$ **by** *auto*

qed

lemma solves-vec-nth:

fixes $f :: ('a :: \text{banach}) ^{('n :: \text{finite})} \Rightarrow ('a ^{'n})$
assumes $(\varphi \text{ solves-ode } (\lambda t. f)) \{0..t\} \text{ UNIV}$
shows $((\lambda t. (\varphi t) \$ i) \text{ solves-ode } (\lambda t s. (f (\varphi t)) \$ i)) \{0..t\} \text{ UNIV}$
using *assms unfolding solves-ode-def has-vderiv-on-def has-vector-derivative-def has-derivative-def*
apply *safe apply* (auto simp: *bounded-linear-def bounded-linear-axioms-def*) [1]
apply (erule-tac $x=x$ **in** *ballE*, *clarsimp*)
apply (rule *convergences-solves-vec-nth*)
by (simp-all add: *Pi-def*)

lemma solves-vec-lambda:

fixes $f :: ('a :: \text{banach}) ^{('n :: \text{finite})} \Rightarrow ('a ^{'n})$ **and** $\varphi :: \text{real} \Rightarrow ('a ^{'n})$
assumes $\forall i :: 'n. ((\lambda t. (\varphi t) \$ i) \text{ solves-ode } (\lambda t s. (f (\varphi t)) \$ i)) \{0..t\} \text{ UNIV}$
shows $(\varphi \text{ solves-ode } (\lambda t. f)) \{0..t\} \text{ UNIV}$
using *assms unfolding solves-ode-def has-vderiv-on-def has-vector-derivative-def has-derivative-def*
apply *safe apply* (auto simp: *bounded-linear-def bounded-linear-axioms-def*) [1]

by(*rule Finite-Cartesian-Product.vec-tendstoI*, *simp-all*)

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

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

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

lemma *origin-line-derivative*: $((\cdot) \ a \ \text{has-derivative} \ (\lambda x. x *_R a)) \ (\text{at } x \ \text{within } T)$
using *origin-line-vector-derivative* **unfolding** *has-vector-derivative-def* **by** *simp*

lemma *quadratic-monomial-derivative*:
 $((\lambda t::\text{real}. a \cdot t^2) \ \text{has-derivative} \ (\lambda t. a \cdot (2 \cdot x \cdot t))) \ (\text{at } x \ \text{within } T)$
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-derivative-div*:
 $((\lambda t::\text{real}. a \cdot t^2 / 2) \ \text{has-derivative} \ (\lambda t. a \cdot x \cdot t)) \ (\text{at } x \ \text{within } T)$
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-derivatives*]: $((\lambda t. a \cdot t^2 / 2) \ \text{has-vderiv-on} \ (\cdot) \ a) \ T$
apply(*simp add: has-vderiv-on-def has-vector-derivative-def*, *clarify*)
using *quadratic-monomial-derivative-div* **by** (*simp add: mult-commute-abs*)

lemma *pos-vderiv*[*poly-derivatives*]:
 $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \ \text{has-vderiv-on} \ (\lambda t. a \cdot t + v)) \ T$
apply(*rule-tac f'*= $\lambda x. a \cdot x + v$ **and** *g'1*= $\lambda x. 0$ **in** *derivative-intros*(190))
apply(*rule-tac f'1*= $\lambda x. a \cdot x$ **and** *g'1*= $\lambda x. v$ **in** *derivative-intros*(190))
using *poly-derivatives*(2) **by**(*auto intro: derivative-intros*)

lemma *pos-derivative*:
 $t \in T \implies ((\lambda \tau. a \cdot \tau^2 / 2 + v \cdot \tau + x) \ \text{has-derivative} \ (\lambda x. x *_R (a \cdot t + v)))$
 $(\text{at } t \ \text{within } T)$
using *pos-vderiv* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *simp*

lemma *vel-vderiv*[*poly-derivatives*]: $((\lambda r. a \cdot r + v) \ \text{has-vderiv-on} \ (\lambda t. a)) \ T$
apply(*rule-tac f'1*= $\lambda x. a$ **and** *g'1*= $\lambda x. 0$ **in** *derivative-intros*(190))
unfolding *has-vderiv-on-def* **by**(*auto intro: derivative-eq-intros*)

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

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

declare *origin-line-vector-derivative* [*poly-derivatives*]
and *origin-line-derivative* [*poly-derivatives*]
and *quadratic-monomial-derivative* [*poly-derivatives*]
and *quadratic-monomial-derivative-div* [*poly-derivatives*]
and *pos-derivative* [*poly-derivatives*]

3.3 Vectors, norms and matrices.

3.3.1 Unit vectors and vector norm

lemma *norm-scalar-mult*: $\text{norm } ((c::\text{real}) * s \ x) = |c| \cdot \text{norm } x$
unfolding *norm-vec-def L2-set-def real-norm-def vector-scalar-mult-def* **apply**
simp
apply(*subgoal-tac* $(\sum_{i \in \text{UNIV}} (c \cdot x \ \$ \ i)^2) = |c|^2 \cdot (\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2)$)
apply(*simp add: real-sqrt-mult*)
apply(*simp add: sum-distrib-left*)
by (*meson power-mult-distrib*)

lemma *squared-norm-vec*: $(\text{norm } x)^2 = (\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2)$
unfolding *norm-vec-def L2-set-def* **by** (*simp add: sum-nonneg*)

lemma *sgn-is-unit-vec*: $\text{sgn } x = 1 / \text{norm } x * s \ x$
unfolding *sgn-vec-def scaleR-vec-def* **by**(*simp add: vector-scalar-mult-def divide-inverse-commute*)

lemma *norm-sgn-unit*: $(x::\text{real}^n) \neq 0 \implies \text{norm } (\text{sgn } x) = 1$
proof(*subst sgn-is-unit-vec, unfold norm-vec-def L2-set-def, simp add: power-divide*)
assume $x \neq 0$
have $(\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2 / (\text{norm } x)^2) = 1 / (\text{norm } x)^2 \cdot (\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2)$
by (*simp add: sum-divide-distrib*)
also have $(\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2) = (\text{norm } x)^2$ **by**(*subst squared-norm-vec, simp*)
ultimately show $(\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2 / (\text{sqrt } (\sum_{i \in \text{UNIV}} (x \ \$ \ i)^2))^2) = 1$
using $\langle x \neq 0 \rangle$ **by** *simp*
qed

lemma *norm-matrix-sgn*: $\text{norm } (A * v \ (x::\text{real}^n)) = \text{norm } (A * v \ (\text{sgn } x)) \cdot \text{norm } x$
unfolding *sgn-is-unit-vec vector-scalar-commute norm-scalar-mult* **by** *simp*

lemma *vector-norm-distr-minus*:
fixes $A::('a::\{\text{real-normed-vector, ring-1}\})^n \times^m$
shows $\text{norm } (A * v \ x - A * v \ y) = \text{norm } (A * v \ (x - y))$
by(*subst matrix-vector-mult-diff-distrib, simp*)

3.3.2 Matrix norm

abbreviation $\text{norm}_S (A::\text{real}^{'n'}^{'m'}) \equiv \text{Sup } \{ \text{norm } (A * v \ x) \mid x. \text{norm } x = 1 \}$

lemma *unit-norms-bound*:

fixes $A::\text{real}^{'n::\text{finite}}^{'m::\text{finite}}}$

shows $\text{norm } x = 1 \implies \text{norm } (A * v \ x) \leq \text{norm } ((\chi \ i \ j. |A \$ i \$ j|) * v \ 1)$

proof–

assume $\text{norm } x = 1$

from *this* **have** $\bigwedge j. |x \$ j| \leq 1$

by (*metis component-le-norm-cart*)

then **have** $\bigwedge i \ j. |A \$ i \$ j| \cdot |x \$ j| \leq |A \$ i \$ j| \cdot 1$

using *mult-left-mono* **by** (*simp add: mult-left-le*)

from *this* **have** $\bigwedge i. (\sum_{j \in \text{UNIV}} |A \$ i \$ j| \cdot |x \$ j|)^2 \leq (\sum_{j \in \text{UNIV}} |A \$ i \$ j|)^2$

by (*simp add: power-mono sum-mono sum-nonneg*)

also **have** $\bigwedge i. (\sum_{j \in \text{UNIV}} A \$ i \$ j \cdot x \$ j)^2 \leq (\sum_{j \in \text{UNIV}} |A \$ i \$ j \cdot x \$ j|)^2$

using *abs-le-square-iff* **by** *force*

moreover **have** $\bigwedge i. (\sum_{j \in \text{UNIV}} |A \$ i \$ j \cdot x \$ j|)^2 = (\sum_{j \in \text{UNIV}} |A \$ i \$ j| \cdot |x \$ j|)^2$

by (*simp add: abs-mult*)

ultimately **have** $\bigwedge i. (\sum_{j \in \text{UNIV}} A \$ i \$ j \cdot x \$ j)^2 \leq (\sum_{j \in \text{UNIV}} |A \$ i \$ j|)^2$

using *order-trans* **by** *fastforce*

hence $(\sum_{i \in \text{UNIV}} (\sum_{j \in \text{UNIV}} A \$ i \$ j \cdot x \$ j)^2) \leq (\sum_{i \in \text{UNIV}} (\sum_{j \in \text{UNIV}} |A \$ i \$ j|)^2)$

by (*simp add: sum-mono*)

then **have** $(\text{sqrt } (\sum_{i \in \text{UNIV}} (\sum_{j \in \text{UNIV}} A \$ i \$ j \cdot x \$ j)^2)) \leq (\text{sqrt } (\sum_{i \in \text{UNIV}} (\sum_{j \in \text{UNIV}} |A \$ i \$ j|)^2))$

using *real-sqrt-le-mono* **by** *blast*

thus $\text{norm } (A * v \ x) \leq \text{norm } ((\chi \ i \ j. |A \$ i \$ j|) * v \ 1)$

by (*simp add: norm-vec-def L2-set-def matrix-vector-mult-def*)

qed

lemma *unit-norms-exists*:

fixes $A::\text{real}^{'n::\text{finite}}^{'m::\text{finite}}}$

shows *bounded:bounded* $\{ \text{norm } (A * v \ x) \mid x. \text{norm } x = 1 \}$

and *bdd-above:bdd-above* $\{ \text{norm } (A * v \ x) \mid x. \text{norm } x = 1 \}$

and *non-empty*: $\{ \text{norm } (A * v \ x) \mid x. \text{norm } x = 1 \} \neq \{ \}$ (**is** $?U \neq \{ \}$)

proof–

show *bounded* $?U$

apply (*unfold bounded-def, rule-tac x=0 in exI, simp add: dist-real-def*)

apply (*rule-tac x=norm ((\chi \ i \ j. |A \\$ i \\$ j|) * v \ 1) in exI, clarsimp*)

using *unit-norms-bound* **by** *blast*

next

show *bdd-above* $?U$

apply (*unfold bdd-above-def, rule-tac x=norm ((\chi \ i \ j. |A \\$ i \\$ j|) * v \ 1) in exI, clarsimp*)

using *unit-norms-bound* **by** *blast*

```

next
  have  $\bigwedge k::'n. \text{norm } (\text{axis } k \ (1::\text{real})) = 1$ 
    using norm-axis-1 by blast
  hence  $\bigwedge k::'n. \text{norm } ((A::\text{real}^{('n::\text{finite})} \wedge 'm) * v \ (\text{axis } k \ (1::\text{real}))) \in ?U$ 
    by blast
  thus  $?U \neq \{\}$  by blast
qed

lemma unit-norms:  $\text{norm } x = 1 \implies \text{norm } (A * v \ x) \leq \text{norm}_S \ A$ 
  using cSup-upper mem-Collect-eq unit-norms-exists(2) by (metis (mono-tags, lifting))

lemma unit-norms-ge-0:  $0 \leq \text{norm}_S \ A$ 
  using ex-norm-eq-1 norm-ge-zero unit-norms basic-trans-rules(23) by blast

lemma norm-sgn-le-norms:  $\text{norm } (A * v \ \text{sgn } x) \leq \text{norm}_S \ A$ 
  apply(cases x=0)
  using sgn-zero unit-norms-ge-0 apply force
  using norm-sgn-unit unit-norms by blast

abbreviation entries  $(A::\text{real}^{('n \wedge 'm)}) \equiv \{A \ \$ \ i \ \$ \ j \mid i \ j. \ i \in (UNIV::'m \ \text{set}) \wedge j \in (UNIV::'n \ \text{set})\}$ 
abbreviation maxAbs  $(A::\text{real}^{('n \wedge 'm)}) \equiv \text{Max } (\text{abs } ` \ (\text{entries } A))$ 

lemma maxAbs-def:  $\text{maxAbs } (A::\text{real}^{('n \wedge 'm)}) = \text{Max } \{ |A \ \$ \ i \ \$ \ j| \mid i \ j. \ i \in (UNIV::'m \ \text{set}) \wedge j \in (UNIV::'n \ \text{set}) \}$ 
  apply(simp add: image-def, rule arg-cong[of - - Max])
  by auto

lemma finite-matrix-abs:
  fixes  $A::\text{real}^{('n::\text{finite}) \wedge ('m::\text{finite})}$ 
  shows finite  $\{|A \ \$ \ i \ \$ \ j| \mid i \ j. \ i \in (UNIV::'m \ \text{set}) \wedge j \in (UNIV::'n \ \text{set})\}$  (is finite ?X)
proof-
  {fix  $i::'m$ 
    have finite  $\{|A \ \$ \ i \ \$ \ j| \mid j. \ j \in (UNIV::'n \ \text{set})\}$ 
      using finite-Atleast-Atmost-nat by fastforce
    hence  $\forall i::'m. \text{finite } \{|A \ \$ \ i \ \$ \ j| \mid j. \ j \in (UNIV::'n \ \text{set})\}$  by blast
    then have finite  $(\bigcup i \in UNIV. \{|A \ \$ \ i \ \$ \ j| \mid j. \ j \in (UNIV::'n \ \text{set})\})$  (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 maxAbs-ge-0:  $\text{maxAbs } A \geq 0$ 
proof-
  have  $\bigwedge i \ j. |A \ \$ \ i \ \$ \ j| \geq 0$  by simp
  also have  $\bigwedge i \ j. \text{maxAbs } A \geq |A \ \$ \ i \ \$ \ j|$ 

```

unfolding *maxAbs-def* using *finite-matrix-abs* *Max-ge maxAbs-def* by *blast*
 finally show $0 \leq \text{maxAbs } A$.
 qed

lemma *norms-le-dims-maxAbs*:
 fixes $A :: \text{real}^{('n :: \text{finite})} \times ('m :: \text{finite})$
 shows $\text{norm}_S A \leq \text{real CARD}('n) \cdot \text{real CARD}('m) \cdot (\text{maxAbs } A)$ (is $\text{norm}_S A$
 $\leq ?n \cdot ?m \cdot (\text{maxAbs } A)$)
proof –
 {fix $x :: (\text{real}, 'n) \text{ vec}$ assume $\text{norm } x = 1$
 hence $\text{comp-le-1} : \forall i :: 'n. |x \$ i| \leq 1$
 by (*simp add: norm-bound-component-le-cart*)
 have $A * v x = (\sum i \in \text{UNIV}. x \$ i * s \text{ column } i A)$
 using *matrix-mult-sum* by *blast*
 hence $\text{norm } (A * v x) \leq (\sum (i :: 'n) \in \text{UNIV}. \text{norm } (x \$ i * s \text{ column } i A))$
 by (*simp add: sum-norm-le*)
 also have $\dots = (\sum (i :: 'n) \in \text{UNIV}. |x \$ i| \cdot \text{norm } (\text{column } i A))$
 by (*simp add: norm-scalar-mult*)
 also have $\dots \leq (\sum (i :: 'n) \in \text{UNIV}. \text{norm } (\text{column } i A))$
 by (*metis (no-types, lifting) Groups.mult-ac(2) comp-le-1 mult-left-le norm-ge-zero sum-mono*)
 also have $\dots \leq (\sum (i :: 'n) \in \text{UNIV}. ?m \cdot \text{maxAbs } A)$
proof (*unfold norm-vec-def L2-set-def real-norm-def*)
 have $\bigwedge i j. |\text{column } i A \$ j| \leq \text{maxAbs } A$
 using *finite-matrix-abs* *Max-ge* unfolding *column-def maxAbs-def* by (*simp, blast*)
 hence $\bigwedge i j. |\text{column } i A \$ j|^2 \leq (\text{maxAbs } A)^2$
 by (*metis (no-types, lifting) One-nat-def abs-ge-zero numerals(2) order-trans-rules(23)*)
 power2-abs power2-le-iff-abs-le
 then have $\bigwedge i. (\sum j \in \text{UNIV}. |\text{column } i A \$ j|^2) \leq (\sum (j :: 'm) \in \text{UNIV}. (\text{maxAbs } A)^2)$
 by (*meson sum-mono*)
 also have $(\sum (j :: 'm) \in \text{UNIV}. (\text{maxAbs } A)^2) = ?m \cdot (\text{maxAbs } A)^2$ by *simp*
 ultimately have $\bigwedge i. (\sum j \in \text{UNIV}. |\text{column } i A \$ j|^2) \leq ?m \cdot (\text{maxAbs } A)^2$
 by *force*
 hence $\bigwedge i. \text{sqrt } (\sum j \in \text{UNIV}. |\text{column } i A \$ j|^2) \leq \text{sqrt } (?m \cdot (\text{maxAbs } A)^2)$
 by (*simp add: real-sqrt-le-mono*)
 also have $\text{sqrt } (?m \cdot (\text{maxAbs } A)^2) \leq \text{sqrt } ?m \cdot \text{maxAbs } A$
 using *maxAbs-ge-0 real-sqrt-mult* by *auto*
 also have $\dots \leq ?m \cdot \text{maxAbs } A$
 using *sqrt-real-nat-le maxAbs-ge-0 mult-right-mono* by *blast*
 finally show $(\sum i \in \text{UNIV}. \text{sqrt } (\sum j \in \text{UNIV}. |\text{column } i A \$ j|^2)) \leq (\sum (i :: 'n) \in \text{UNIV}. ?m \cdot \text{maxAbs } A)$
 by (*meson sum-mono*)
 qed
 also have $(\sum (i :: 'n) \in \text{UNIV}. (\text{maxAbs } A)) = ?n \cdot (\text{maxAbs } A)$
 using *sum-constant-scale* by *auto*
 ultimately have $\text{norm } (A * v x) \leq ?n \cdot ?m \cdot (\text{maxAbs } A)$ by *simp*}

```

    from this show ?thesis
    using unit-norms-exists[of A] Connected.bounded-has-Sup(2) by blast
qed

end
theory cat2rel
  imports cat2rel-pre

begin

```

4 Hybrid System Verification

4.1 Verification by providing solutions

```

abbreviation orbital f T S t0 x0 ≡
  {x t | t x. t ∈ T ∧ (x solves-ode f) T S ∧ x t0 = x0 ∧ x0 ∈ S ∧ t0 ∈ T}
abbreviation g-orbital f T S t0 x0 G ≡
  {x t | t x. t ∈ T ∧ (x solves-ode f) T S ∧ x t0 = x0 ∧ x0 ∈ S ∧ t0 ∈ T ∧ (∀ r
    ∈ {t0--t}. G (x r))}

abbreviation
g-evolution :: (real ⇒ ('a::banach) ⇒ 'a) ⇒ real set ⇒ 'a set ⇒ real ⇒ 'a pred
⇒ 'a rel
((1 { [x'=-]- - @ - & -})) where {[x'=f] T S @ t0 & G} ≡  $\mathcal{R} (\lambda s. g\text{-orbital } f \text{ } T \text{ } S \text{ } t0 \text{ } s \text{ } G)$ 

```

```

context picard-ivp
begin

```

```

lemma orbital-collapses:
  assumes ivp:  $\forall s \in S. ((\lambda t. \varphi \text{ } t \text{ } s) \text{ solves-ode } f) T S \wedge \varphi \text{ } t0 \text{ } s = s$  and  $s \in S$ 
  shows orbital f T S t0 s = { $\varphi \text{ } t \text{ } s \mid t. t \in T$ }
  apply safe apply (rule-tac x=t in exI, simp)
  apply (rule-tac x=xa and s=xa t0 in unique-solution, simp-all add: assms)
  apply (rule-tac x=t in exI, rule-tac x= $\lambda t. \varphi \text{ } t \text{ } s$  in exI)
  using assms init-time by auto

```

```

lemma g-orbital-collapses:
  assumes ivp:  $\forall s \in S. ((\lambda t. \varphi \text{ } t \text{ } s) \text{ solves-ode } f) T S \wedge \varphi \text{ } t0 \text{ } s = s$  and  $s \in S$ 
  shows g-orbital f T S t0 s G = { $\varphi \text{ } t \text{ } s \mid t. t \in T \wedge (\forall r \in \{t0--t\}. G (\varphi \text{ } r \text{ } s))$ }
  apply safe apply (rule-tac x=t in exI, simp)
  using assms unique-solution apply (metis closed-segment-subset-domainI)
  apply (rule-tac x=t in exI, rule-tac x= $\lambda t. \varphi \text{ } t \text{ } s$  in exI)
  using assms init-time by auto

```

```

lemma wp-orbit:
  assumes ivp:  $\forall s \in S. ((\lambda t. \varphi \text{ } t \text{ } s) \text{ solves-ode } f) T S \wedge \varphi \text{ } t0 \text{ } s = s$ 
  shows wp ( $\mathcal{R} (\lambda s. \text{orbital } f \text{ } T \text{ } S \text{ } t0 \text{ } s)$ )  $\lceil Q \rceil = \lceil \lambda s. \forall t \in T. s \in S \longrightarrow Q (\varphi \text{ } t \text{ } s) \rceil$ 

```

```

apply(subst wp-rel, simp add: f2r-def, safe)
apply(erule-tac x= $\varphi$  t s in allE, erule impE)
apply(rule-tac x=t in exI, rule-tac x= $\lambda$  t.  $\varphi$  t s in exI)
using ivp init-time apply(simp, simp)
apply(subgoal-tac  $\varphi$  t (x t0) = x t)
apply(erule-tac x=t in ballE, simp, simp)
by(rule-tac y=x and s=x t0 in unique-solution, simp-all add: assms)

```

lemma wp-g-orbit:

```

assumes ivp: $\forall s \in S. ((\lambda t. \varphi$  t s) solves-ode f) T S  $\wedge \varphi$  t0 s = s
shows wp {[x'=f] T S @ t0 & G} [Q] = [ $\lambda$  s.  $\forall t \in T. s \in S \longrightarrow (\forall r \in$ 
{t0--t}. G ( $\varphi$  r s))  $\longrightarrow Q$  ( $\varphi$  t s)]
apply(subst wp-rel, simp add: f2r-def, safe)
apply(erule-tac x= $\varphi$  t s in allE, erule impE)
apply(rule-tac x=t in exI, rule-tac x= $\lambda$  t.  $\varphi$  t s in exI)
apply(simp add: ivp init-time, simp)
apply(subgoal-tac  $\forall r \in \{t0--t\}. \varphi$  r (x t0) = x r)
apply(erule-tac x=t in ballE, safe)
apply(erule-tac x=r in ballE)+ apply simp-all
apply(erule-tac x=t in ballE)+ apply simp-all
apply(rule-tac y=x and s=x t0 in unique-solution, simp-all add: assms)
using subsegment by blast

```

end

lemma dSolution:

```

assumes picard-ivp f T S L t0 and ivp: $\forall s \in S. ((\lambda t. \varphi$  t s) solves-ode f) T S  $\wedge$ 
 $\varphi$  t0 s = s
and  $\forall s. P$  s  $\longrightarrow (\forall t \in T. s \in S \longrightarrow (\forall r \in \{t0..t\}. G$  ( $\varphi$  r s))  $\longrightarrow Q$  ( $\varphi$  t
s))
shows [P]  $\subseteq$  wp {[x'=f] T S @ t0 & G} [Q]
using assms apply(subst picard-ivp.wp-g-orbit, auto)
by (simp add: Starlike.closed-segment-eq-real-ivl)

```

This last theorem allows us to compute weakest liberal preconditions for known systems of ODEs:

```

corollary line-DS:  $0 \leq t \implies$  wp {[x'= $\lambda$  t s. c]{0..t} UNIV @ 0 & G} [Q] =
[ $\lambda$  x.  $\forall \tau \in \{0..t\}. (\forall r \in \{0--\tau\}. G$  (x + r *R c))  $\longrightarrow Q$  (x +  $\tau$  *R c)]
apply(subst picard-ivp.wp-g-orbit[of  $\lambda$  t s. c - 1/(t + 1) - ( $\lambda$  t x. x + t *R c)])
using constant-is-picard-ivp apply blast
using line-solves-constant by auto

```

4.2 Verification with differential invariants

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

we prove the inference rules which are used in verification proofs.

4.2.1 Differential Weakening

theorem *DW*:

shows $wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil Q \rceil = wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil \lambda s. G \ s \longrightarrow Q \ s \rceil$
unfolding *rel-antidomain-kleene-algebra.fbox-def rel-ad-def f2r-def*
apply(*simp add: relcomp.simps p2r-def*)
apply(*rule subset-antisym*)
by *fastforce+*

theorem *dWeakening*:

assumes $\lceil G \rceil \subseteq \lceil Q \rceil$
shows $\lceil P \rceil \subseteq wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil Q \rceil$
using *assms* **apply**(*subst wp-rel*)
by(*auto simp: f2r-def*)

4.2.2 Differential Cut

lemma *wp-g-orbit-IdD*:

assumes $wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil C \rceil = Id$ **and** $\forall r \in \{t0 \dashv\dashv t\}. (a, x \ r) \in \{[x'=f] T S @ t0 \ \& \ G\}$
shows $\forall r \in \{t0 \dashv\dashv t\}. C \ (x \ r)$

proof–

{fix $r :: real$
have $\bigwedge R \ P \ s. wp \ R \ \lceil P \rceil \neq Id \vee (\forall y. (s::'a, y) \in R \longrightarrow P \ y)$
by (*metis (lifting) p2r-IdD wp-rel*)
then have $r \notin \{t0 \dashv\dashv t\} \vee C \ (x \ r)$ **using** *assms* **by** *blast*
then show *?thesis* **by** *blast*

qed

theorem *DC*:

assumes $t0 \in T$ **and** *interval* T
and $wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil C \rceil = Id$
shows $wp \ (\{[x'=f] T S @ t0 \ \& \ G\}) \ \lceil Q \rceil = wp \ \{[x'=f] T S @ t0 \ \& \ \lambda s. G \ s \wedge C \ s\} \ \lceil Q \rceil$

proof(*rule-tac f= $\lambda x. wp \ x \ \lceil Q \rceil$ in HOL.arg-cong, safe*)

fix $a \ b$ **assume** $(a, b) \in \{[x'=f] T S @ t0 \ \& \ G\}$
then obtain $t::real$ **and** x **where** $t \in T$ **and** $x \text{-solves}:(x \text{ solves-ode } f) \ T \ S$ **and**

$x \ t0 = a$ **and** $guard\text{-}x:(\forall r \in \{t0 \dashv\dashv t\}. G \ (x \ r))$ **and** $a \in S$ **and** $b = x \ t$

unfolding *f2r-def* **by** *blast*

from *guard-x* **have** $\forall r \in \{t0 \dashv\dashv t\}. \forall \tau \in \{t0 \dashv\dashv r\}. G \ (x \ \tau)$

using *assms(1)* **by** (*metis contra-subsetD ends-in-segment(1) subset-segment(1)*)

also have $\forall r \in \{t0 \dashv\dashv t\}. r \in T$

using *assms(1,2)* $\langle t \in T \rangle$ *interval.closed-segment-subset-domain* **by** *blast*

ultimately have $\forall r \in \{t0 \dashv\dashv t\}. (a, x \ r) \in \{[x'=f] T S @ t0 \ \& \ G\}$

using $x\text{-solves } \langle x \ t0 = a \rangle \langle a \in S \rangle$ **unfolding** $f2r\text{-def}$ **by** *blast*
 from *this* have $\forall r \in \{t0 \dashv\dashv t\}. C \ (x \ r)$ using $wp\text{-g-orbit-IdD } assms(3)$ **by** *blast*
 thus $(a, b) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ \lambda s. G \ s \wedge C \ s\}$ **unfolding** $f2r\text{-def}$
 using $guard\text{-}x \ \langle a \in S \rangle \langle b = x \ t \rangle \langle t \in T \rangle \langle x \ t0 = a \rangle$ $x\text{-solves } \langle \forall r \in \{t0 \dashv\dashv t\}. r \in T \rangle$ **by** *fastforce*
next
 fix $a \ b$ **assume** $(a, b) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ \lambda s. G \ s \wedge C \ s\}$
 then **show** $(a, b) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ G\}$
 unfolding $f2r\text{-def}$ **by** *blast*
qed

theorem $dCut$:

assumes $t0 \in T$ **and** *interval* T
 and $wp\text{-}C: [P] \subseteq wp \ (\{[x'=f] T \ S \ @ \ t0 \ \& \ G\}) \ [C]$
 and $wp\text{-}Q: [P] \subseteq wp \ (\{[x'=f] T \ S \ @ \ t0 \ \& \ (\lambda s. G \ s \wedge C \ s)\}) \ [Q]$
 shows $[P] \subseteq wp \ (\{[x'=f] T \ S \ @ \ t0 \ \& \ G\}) \ [Q]$
proof(*subst wp-rel, simp add: p2r-def, clarsimp*)
 fix $a \ y$ **assume** $P \ a$ **and** $(a, y) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ G\}$
 then **obtain** $x \ t$ **where** $t \in T$ **and** $x\text{-solves}:(x \text{ solves-ode } f) T \ S$ **and** $x \ t = y$
 and $x \ t0 = a$ **and** $guard\text{-}x: (\forall r \in \{t0 \dashv\dashv t\}. G \ (x \ r))$ **and** $a \in S$ **by**(*auto simp: f2r-def*)
 from $guard\text{-}x$ have $\forall r \in \{t0 \dashv\dashv t\}. \forall \tau \in \{t0 \dashv\dashv r\}. G \ (x \ \tau)$
 using $assms(1)$ **by** (*metis contra-subsetD ends-in-segment(1) subset-segment(1)*)

 also have $\forall r \in \{t0 \dashv\dashv t\}. r \in T$
 using $assms(1,2)$ $\langle t \in T \rangle$ *interval.closed-segment-subset-domain* **by** *blast*
 ultimately have $\forall r \in \{t0 \dashv\dashv t\}. (a, x \ r) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ G\}$
 using $x\text{-solves } \langle x \ t0 = a \rangle \langle a \in S \rangle$ **unfolding** $f2r\text{-def}$ **by** *blast*
 from *this* have $\forall r \in \{t0 \dashv\dashv t\}. C \ (x \ r)$ using $assms(3)$ $\langle P \ a \rangle$ **by**(*subst (asm) wp-rel*) *auto*
 hence $(a, y) \in \{[x'=f] T \ S \ @ \ t0 \ \& \ \lambda s. G \ s \wedge C \ s\}$ **unfolding** $f2r\text{-def}$
 using $guard\text{-}x \ \langle a \in S \rangle \langle x \ t = y \rangle \langle t \in T \rangle \langle x \ t0 = a \rangle$ $x\text{-solves } \langle \forall r \in \{t0 \dashv\dashv t\}. r \in T \rangle$ **by** *fastforce*
 from *this* $\langle P \ a \rangle$ **and** $wp\text{-}Q$ **show** $Q \ y$
by(*subst (asm) wp-rel, simp add: f2r-def*)
qed

corollary $dCut\text{-interval}$:

assumes $t0 \leq t$ **and** $[P] \subseteq wp \ (\{[x'=f] \{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ [C]$
 and $[P] \subseteq wp \ (\{[x'=f] \{t0..t\} \ S \ @ \ t0 \ \& \ (\lambda s. G \ s \wedge C \ s)\}) \ [Q]$
 shows $[P] \subseteq wp \ (\{[x'=f] \{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ [Q]$
apply(*rule-tac C=C in dCut*)
 using $assms$ **by**(*simp-all add: interval-def*)

4.2.3 Differential Invariant

lemma $DI\text{-sufficiency}$:

assumes $picard: picard\text{-ivp } f \ T \ S \ L \ t0$
 shows $wp \ \{[x'=f] T \ S \ @ \ t0 \ \& \ G\} \ [Q] \subseteq wp \ [G] \ [\lambda s. s \in S \longrightarrow Q \ s]$

proof(subst wp-rel, subst wp-rel, simp add: p2r-def, clarsimp)
 fix s assume wlpQ: $\forall y. (s, y) \in \{[x'=f]T S @ t0 \ \& \ G\} \longrightarrow Q \ y$ and s $\in S$
 and G s
 from this and picard obtain x where (x solves-ode f) T S \wedge x t0 = s
 using picard-ivp.fixed-point-solves by blast
 then also have $\forall r \in \{t0 \dots t0\}. G \ (x \ r)$ using $\langle G \ s \rangle$ by simp
 ultimately have (s,s) $\in \{[x'=f]T S @ t0 \ \& \ G\}$
 using picard picard-ivp.init-time $\langle s \in S \rangle$ f2r-def by fastforce
 thus Q s using wlpQ by blast
qed

definition pderivative :: 'a pred \Rightarrow 'a pred \Rightarrow (real \Rightarrow ('a::real-normed-vector) \Rightarrow 'a) \Rightarrow real set \Rightarrow
 'a set \Rightarrow bool ((-)/ is'-pderivative'-of (-)/ with'-respect'-to (-) (-) (-) [70, 65] 61)
 where
 I' is-pderivative-of I with-respect-to f T S \equiv bdd-below T \wedge ($\forall x. (x \text{ solves-ode } f) T S \longrightarrow$
 I (x (Inf T)) \longrightarrow ($\forall t \in T. (\forall r \in \{(Inf T) \dots t\}. I' (x \ r)) \longrightarrow (I (x \ t)))$)

lemma dInvariant:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
 assumes $\lceil G \rceil \subseteq \lceil I' \rceil$ and I' is-pderivative-of I with-respect-to f T S
 shows $\lceil I \rceil \subseteq \text{wp} (\{[x'=f]T S @ (Inf T) \ \& \ G\}) \lceil I \rceil$
 using assms unfolding pderivative-def apply(subst wp-rel)
proof(simp add: p2r-def, clarsimp)
 assume prime: $\forall x. (x \text{ solves-ode } f) T S \longrightarrow I (x (Inf T)) \longrightarrow (\forall t \in T. (\forall r \in \{(Inf T) \dots t\}. I' (x \ r)) \longrightarrow I (x \ t))$
 fix s y assume (s,y) $\in \{[x'=f]T S @ (Inf T) \ \& \ G\}$ and sHyp: I s and bdd-below T
 then obtain x and t where x-ivp: (x solves-ode f) T S \wedge x (Inf T) = s
 and xtHyp: x t = y \wedge t $\in T$ and GHyp: $\forall r \in \{(Inf T) \dots t\}. G \ (x \ r)$
 by (simp add: f2r-def, clarify, auto)
 hence (Inf T) $\leq t$ by (simp add: $\langle \text{bdd-below } T \rangle$ cInf-lower)
 from GHyp and $\langle \lceil G \rceil \subseteq \lceil I' \rceil \rangle$ have geq0: $\forall r \in \{(Inf T) \dots t\}. I' (x \ r)$
 by (auto simp: p2r-def)
 thus I y using xtHyp x-ivp sHyp and prime by blast
qed

lemma invariant-eq-0:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
 assumes nuHyp: $\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf T) \dots t\}. ((\lambda \tau. \vartheta (x \ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_{\mathbb{R}} \nu (x \ r))) \text{ (at } r \text{ within } \{(Inf T) \dots t\}))$
 and $\lceil G \rceil \subseteq \lceil \lambda s. \nu s = 0 \rceil$ and bdd-below T
 shows $\lceil \lambda s. \vartheta s = 0 \rceil \subseteq \text{wp} (\{[x'=f]T S @ (Inf T) \ \& \ G\}) \lceil \lambda s. \vartheta s = 0 \rceil$
 apply(rule dInvariant [of - $\lambda s. \nu s = 0$])
 unfolding pderivative-def using assms apply(simp, simp)
proof(clarify)
 fix x and t
 assume x-ivp: (x solves-ode f) T S \wedge $\vartheta (x (Inf T)) = 0$

and $tHyp: t \in T$ **and** $eq0: \forall r \in \{Inf\ T - - t\}. \nu\ (x\ r) = 0$
hence $(Inf\ T) \leq t$ **by** (*simp add: <bdd-below T> cInf-lower*)
have $\forall r \in \{(Inf\ T) - - t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_R \nu\ (x\ r)))$
(at r within {(Inf T) - - t}) **using** *nuHyp x-ivp(1)* **and** *tHyp* **by** *auto*
then have $\forall r \in \{(Inf\ T) - - t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_R 0))$
(at r within {(Inf T) - - t}) **using** *eq0* **by** *auto*
then have $\exists r \in \{(Inf\ T) - - t\}. \vartheta\ (x\ t) - \vartheta\ (x\ (Inf\ T)) = (\lambda\tau. \tau *_R 0)\ (t - (Inf\ T))$
by(*rule-tac closed-segment-mvt, auto simp: <(Inf T) ≤ t>*)
thus $\vartheta\ (x\ t) = 0$
using *x-ivp(2)* **by** (*metis right-minus-eq scale-zero-right*)
qed

corollary *invariant-eq-0-interval:*

fixes $\vartheta::'a::banach \Rightarrow real$
assumes $\forall x. (x\ solves-ode\ f)\{t0..t\}\ S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
and $\lceil G \rceil \subseteq \lceil \lambda s. \nu\ s = 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta\ s = 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \& \ G\})\ \lceil \lambda s. \vartheta\ s = 0 \rceil$
apply(*subgoal-tac* $\lceil \lambda s. \vartheta\ s = 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S @ (Inf\ \{t0..t\}) \ \& \ G\})$
 $\lceil \lambda s. \vartheta\ s = 0 \rceil$)
apply(*subgoal-tac* $Inf\ \{t0..t\} = t0$, *simp*)
using $\langle t0 \leq t \rangle$ **apply** *simp*
apply(*rule invariant-eq-0[of - {t0..t} - - ν]*)
using *assms* **by**(*auto simp: closed-segment-eq-real-ivl*)

theorem *dInvariant-eq-0:*

fixes $\vartheta::'a::banach \Rightarrow real$ **and** $\nu::'a \Rightarrow real$
assumes $\forall x. (x\ solves-ode\ f)\{t0..t\}\ S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_R \nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
and *impls:* $\lceil P \rceil \subseteq \lceil \lambda s. \vartheta\ s = 0 \rceil$ $\lceil \lambda s. \vartheta\ s = 0 \rceil \subseteq \lceil Q \rceil$ $\lceil G \rceil \subseteq \lceil \lambda s. \nu\ s = 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \& \ G\})\ \lceil Q \rceil$
apply(*rule-tac* $C = \lambda s. \vartheta\ s = 0$ **in** *dCut-interval, simp add: <t0 ≤ t>*)
apply(*subgoal-tac* $\lceil \lambda s. \vartheta\ s = 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \& \ G\})\ \lceil \lambda s. \vartheta\ s = 0 \rceil$)
using *impls* **apply** *blast*
apply(*rule-tac* $\nu = \nu$ **in** *invariant-eq-0-interval*)
using *assms(1,4,5)* **apply**(*simp, simp, simp*)
apply(*rule dWeakening*)
using *impls* **by** *simp*

lemma *invariant-geq-0:*

fixes $\vartheta::'a::banach \Rightarrow real$
assumes *nuHyp:* $\forall x. (x\ solves-ode\ f)\ T\ S \longrightarrow (\forall t \in T. \forall r \in \{(Inf\ T) - - t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{(Inf\ T) - - t\}))$
and $\lceil G \rceil \subseteq \lceil \lambda s. (\nu\ s) \geq 0 \rceil$ **and** *bdd-below T*
shows $\lceil \lambda s. \vartheta\ s \geq 0 \rceil \subseteq wp\ (\{[x'=f]\ T\ S @ (Inf\ T) \ \& \ G\})\ \lceil \lambda s. \vartheta\ s \geq 0 \rceil$

apply(rule *dInvariant* [of - $\lambda s. \nu s \geq 0$])
unfolding *pderivative-def* **using** *assms* **apply**(*simp*, *simp*)
proof(*clarify*)
fix *x* **and** *t*
assume *x-ivp*: $\vartheta (x (Inf T)) \geq 0$ (*x solves-ode f*) *T S*
and *tHyp*: $t \in T$ **and** *ge0*: $\forall r \in \{Inf T - t\}. \nu (x r) \geq 0$
hence $(Inf T) \leq t$ **by** (*simp add*: $\langle bdd\text{-}below\ T \rangle$ *cInf-lower*)
have $\forall r \in \{(Inf T) - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r))))$
(at *r* within $\{(Inf T) - t\}$) **using** *nuHyp x-ivp(2)* **and** *tHyp* **by** *auto*
then have $\exists r \in \{(Inf T) - t\}. \vartheta (x t) - \vartheta (x (Inf T)) = (\lambda \tau. \tau *_R (\nu (x r))) (t - (Inf T))$
by(rule-tac *closed-segment-mvt*, *auto simp*: $\langle Inf\ T \leq t \rangle$)
from this obtain *r* **where**
 $r \in \{(Inf T) - t\} \wedge \vartheta (x t) = (t - Inf\ T) *_R \nu (x r) + \vartheta (x (Inf\ T))$ **by force**

thus $0 \leq \vartheta (x t)$ **by** (*simp add*: $\langle Inf\ T \leq t \rangle$ *ge0 x-ivp(1)*)
qed

corollary *invariant-geq-0-interval*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r)))) (at\ r\ within\ \{t0..\tau\}))$
and $\lceil G \rceil \subseteq \lceil \lambda s. \nu s \geq 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta s \geq 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \geq 0 \rceil$
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \geq 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\} S @ (Inf\ \{t0..t\}) \ \& \ G\})$
 $\lceil \lambda s. \vartheta s \geq 0 \rceil$)
apply(*subgoal-tac* $Inf\ \{t0..t\} = t0$, *simp*)
using $\langle t0 \leq t \rangle$ **apply**(*simp add*: *closed-segment-eq-real-ivl*)
apply(rule *invariant-geq-0*[of - $\{t0..t\}$ - ν])
using *assms* **by**(*auto simp*: *closed-segment-eq-real-ivl*)

theorem *dInvariant-geq-0*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$ **and** $\nu :: 'a \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x r)))) (at\ r\ within\ \{t0..\tau\}))$
and *impls*: $\lceil P \rceil \subseteq \lceil \lambda s. \vartheta s \geq 0 \rceil$ $\lceil \lambda s. \vartheta s \geq 0 \rceil \subseteq \lceil Q \rceil$ $\lceil G \rceil \subseteq \lceil \lambda s. \nu s \geq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \subseteq wp\ (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil Q \rceil$
apply(rule-tac $C = \lambda s. \vartheta s \geq 0$ **in** *dCut-interval*, *simp add*: $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \geq 0 \rceil \subseteq wp\ (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \geq 0 \rceil$)
using *impls* **apply** *blast*
apply(rule-tac $\nu = \nu$ **in** *invariant-geq-0-interval*)
using *assms*(1,4,5) **apply**(*simp*, *simp*, *simp*)
apply(rule *dWeakening*)
using *impls* **by** *simp*

lemma *invariant-leq-0*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $\text{nuHyp}:\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf\ T)---t\}. ((\lambda\tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x\ r)))) (at\ r\ \text{within } \{(Inf\ T)---t\}))$
and $\lceil G \rceil \subseteq \lceil \lambda s. (\nu\ s) \leq 0 \rceil$ **and** $\text{bdd-below } T$
shows $\lceil \lambda s. \vartheta\ s \leq 0 \rceil \subseteq \text{wp } (\{[x'=f] T S @ (Inf\ T) \ \& \ G\}) \lceil \lambda s. \vartheta\ s \leq 0 \rceil$
apply($\text{rule } dInvariant \text{ [of - } \lambda\ s. \nu\ s \leq 0]$)
unfolding pderivative-def **using** assms **apply**(simp, simp)
proof(clarify)
fix x **and** t
assume $x\text{-ivp}:\vartheta (x (Inf\ T)) \leq 0 (x \text{ solves-ode } f) T S$
and $tHyp:t \in T$ **and** $ge0:\forall r \in \{Inf\ T---t\}. \nu (x\ r) \leq 0$
hence $(Inf\ T) \leq t$ **by** ($\text{simp add: } \langle \text{bdd-below } T \rangle \text{ cInf-lower}$)
have $\forall r \in \{(Inf\ T)---t\}. ((\lambda\tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x\ r))))$
(at r **within** $\{(Inf\ T)---t\}$ **using** $\text{nuHyp } x\text{-ivp}(2)$ **and** $tHyp$ **by** auto
then have $\exists r \in \{(Inf\ T)---t\}. \vartheta (x\ t) - \vartheta (x (Inf\ T)) = (\lambda\tau. \tau *_R (\nu (x\ r))) (t - (Inf\ T))$
by($\text{rule-tac closed-segment-mvt, auto simp: } \langle (Inf\ T) \leq t \rangle$)
from this obtain r **where**
 $r \in \{(Inf\ T)---t\} \wedge \vartheta (x\ t) = (t - Inf\ T) *_R \nu (x\ r) + \vartheta (x (Inf\ T))$ **by force**
thus $\vartheta (x\ t) \leq 0$ **using** $\langle (Inf\ T) \leq t \rangle ge0\ x\text{-ivp}(1)$
by ($\text{metis add-decreasing2 ge-iff-diff-ge-0 split-scaleR-neg-le}$)
qed

corollary invariant-leq-0-interval:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x\ r)))) (at\ r\ \text{within } \{t0..\tau\}))$
and $\lceil G \rceil \subseteq \lceil \lambda s. \nu\ s \leq 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta\ s \leq 0 \rceil \subseteq \text{wp } (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta\ s \leq 0 \rceil$
apply($\text{subgoal-tac } \lceil \lambda s. \vartheta\ s \leq 0 \rceil \subseteq \text{wp } (\{[x'=f] \{t0..t\} S @ (Inf\ \{t0..t\}) \ \& \ G\}) \lceil \lambda s. \vartheta\ s \leq 0 \rceil$)
apply($\text{subgoal-tac } Inf\ \{t0..t\} = t0, \text{simp}$)
using $\langle t0 \leq t \rangle$ **apply**($\text{simp add: closed-segment-eq-real-ivl}$)
apply($\text{rule invariant-leq-0 [of - } \{t0..t\} \text{ - - } \nu]$)
using assms **by**($\text{auto simp: closed-segment-eq-real-ivl}$)

theorem dInvariant-leq-0:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$ **and** $\nu::'a \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R \nu (x\ r)))) (at\ r\ \text{within } \{t0..\tau\}))$
and $\text{impls:} \lceil P \rceil \subseteq \lceil \lambda s. \vartheta\ s \leq 0 \rceil \lceil \lambda s. \vartheta\ s \leq 0 \rceil \subseteq \lceil Q \rceil \lceil G \rceil \subseteq \lceil \lambda s. \nu\ s \leq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \subseteq \text{wp } (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) \lceil Q \rceil$
apply($\text{rule-tac } C=\lambda s. \vartheta\ s \leq 0$ **in** $dCut\text{-interval, simp add: } \langle t0 \leq t \rangle$)
apply($\text{subgoal-tac } \lceil \lambda s. \vartheta\ s \leq 0 \rceil \subseteq \text{wp } (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta\ s \leq 0 \rceil$)
using impls **apply** blast
apply($\text{rule-tac } \nu=\nu$ **in** $\text{invariant-leq-0-interval}$)

```

using assms(1,4,5) apply(simp, simp, simp)
apply(rule dWeakening)
using impls by simp

lemma invariant-above-0:
  fixes  $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$ 
  assumes  $\text{nuHyp} : \forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf\ T) -- t\}. ((\lambda \tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x\ r)))) (at\ r\ \text{within } \{(Inf\ T) -- t\}))$ 
    and  $\lceil G \rceil \subseteq \lceil \lambda s. (\nu\ s) \geq 0 \rceil$  and bdd-below T
  shows  $\lceil \lambda s. \vartheta\ s > 0 \rceil \subseteq wp\ (\{[x'=f] T S @ (Inf\ T) \ \&\ G\})\ \lceil \lambda s. \vartheta\ s > 0 \rceil$ 
  apply(rule dInvariant [of -  $\lambda s. \nu\ s \geq 0$ ])
  unfolding pderivative-def using assms apply(simp, simp)
proof(clarify)
  fix x and t
  assume  $x\text{-ivp} : (x \text{ solves-ode } f) T S \vartheta (x (Inf\ T)) > 0$ 
    and  $tHyp : t \in T$  and  $ge0 : \forall r \in \{(Inf\ T) -- t\}. \nu (x\ r) \geq 0$ 
  hence  $(Inf\ T) \leq t$  by (simp add: <bdd-below T> cInf-lower)
  have  $\forall r \in \{(Inf\ T) -- t\}. ((\lambda \tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x\ r))))$ 
     $(at\ r\ \text{within } \{(Inf\ T) -- t\})$  using nuHyp x-ivp(1) and tHyp by auto
  then have  $\exists r \in \{(Inf\ T) -- t\}. \vartheta (x\ t) - \vartheta (x (Inf\ T)) = (\lambda \tau. \tau *_R (\nu (x\ r))) (t - (Inf\ T))$ 
    by(rule-tac closed-segment-mvt, auto simp: <(Inf T) ≤ t>)
  from this obtain r where
     $r \in \{(Inf\ T) -- t\} \wedge \vartheta (x\ t) = (t - Inf\ T) *_R \nu (x\ r) + \vartheta (x (Inf\ T))$  by force

  thus  $0 < \vartheta (x\ t)$ 
  by (metis <(Inf T) ≤ t> ge0 x-ivp(2) Groups.add-ac(2) add-mono-thms-linordered-field(3))

  ge-iff-diff-ge-0 monoid-add-class.add-0-right scaleR-nonneg-nonneg
qed

corollary invariant-above-0-interval:
  fixes  $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$ 
  assumes  $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x\ r)))) (at\ r\ \text{within } \{t0..\tau\}))$ 
    and  $\lceil G \rceil \subseteq \lceil \lambda s. \nu\ s \geq 0 \rceil$  and  $t0 \leq t$ 
  shows  $\lceil \lambda s. \vartheta\ s > 0 \rceil \subseteq wp\ (\{[x'=f] \{t0..t\} S @ t0 \ \&\ G\})\ \lceil \lambda s. \vartheta\ s > 0 \rceil$ 
  apply(subgoal-tac  $\lceil \lambda s. \vartheta\ s > 0 \rceil \subseteq wp\ (\{[x'=f] \{t0..t\} S @ (Inf\ \{t0..t\}) \ \&\ G\})\ \lceil \lambda s. \vartheta\ s > 0 \rceil$ )
  apply(subgoal-tac  $Inf\ \{t0..t\} = t0$ , simp)
  using  $t0 \leq t$  apply(simp add: closed-segment-eq-real-ivl)
  apply(rule invariant-above-0 [of -  $\{t0..t\}$  - -  $\nu$ ])
  using assms by(auto simp: closed-segment-eq-real-ivl)

theorem dInvariant-above-0:
  fixes  $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$  and  $\nu :: 'a \Rightarrow \text{real}$ 
  assumes  $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x\ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x\ r)))) (at\ r\ \text{within } \{t0..\tau\})$ 

```

and $\text{impls}:[P] \subseteq [\lambda s. \vartheta s > 0] \ [\lambda s. \vartheta s > 0] \subseteq [Q] \ [G] \subseteq [\lambda s. \nu s \geq 0]$
 and $t0 \leq t$
 shows $[P] \subseteq \text{wp} (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [Q]$
 apply($\text{rule-tac } C=\lambda s. \vartheta s > 0$ in $d\text{Cut-interval}$, $\text{simp add: } \langle t0 \leq t \rangle$)
 apply($\text{subgoal-tac } [\lambda s. \vartheta s > 0] \subseteq \text{wp} (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [\lambda s. \vartheta s > 0]$)
 using impls apply blast
 apply($\text{rule-tac } \nu=\nu$ in $\text{invariant-above-0-interval}$)
 using $\text{assms}(1,4,5)$ apply(simp, simp, simp)
 apply(rule dWeakening)
 using impls by simp

lemma invariant-below-0 :

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
 assumes $\text{nuHyp}:\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf T)---t\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{(Inf T)---t\}))$
 and $[G] \subseteq [\lambda s. (\nu s) \leq 0]$ and $\text{bdd-below } T$
 shows $[\lambda s. \vartheta s < 0] \subseteq \text{wp} (\{[x'=f] T S @ (Inf T) \ \& \ G\}) \ [\lambda s. \vartheta s < 0]$
 apply($\text{rule dInvariant [of - } \lambda s. \nu s \leq 0]$)
 unfolding pderivative-def using $\text{assms apply(simp, simp)}$
proof(clarify)
 fix x and t
 assume $x\text{-ivp}:(x \text{ solves-ode } f) T S \vartheta (x (Inf T)) < 0$
 and $t\text{Hyp}:t \in T$ and $ge0:\forall r \in \{(Inf T)---t\}. \nu (x r) \leq 0$
 hence $(Inf T) \leq t$ by ($\text{simp add: } \langle \text{bdd-below } T \rangle \text{ cInf-lower}$)
 have $\forall r \in \{(Inf T)---t\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{(Inf T)---t\})$ using $\text{nuHyp } x\text{-ivp}(1)$ and $t\text{Hyp}$ by auto
 then have $\exists r \in \{(Inf T)---t\}. \vartheta (x t) - \vartheta (x (Inf T)) = (\lambda\tau. \tau *_R (\nu (x r))) (t - (Inf T))$
 by($\text{rule-tac closed-segment-mvt, auto simp: } \langle (Inf T) \leq t \rangle$)
 thus $\vartheta (x t) < 0$ using $\langle (Inf T) \leq t \rangle \text{ ge0 } x\text{-ivp}(2)$
 by ($\text{metis add-mono-thms-linordered-field}(3) \text{ diff-gt-0-iff-gt ge-iff-diff-ge-0 linorder-not-le}$)
 $\text{monoid-add-class.add-0-left monoid-add-class.add-0-right split-scaleR-neg-le}$

qed

corollary $\text{invariant-below-0-interval}$:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
 assumes $\forall x. (x \text{ solves-ode } f)\{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{t0..\tau\}))$
 and $[G] \subseteq [\lambda s. \nu s \leq 0]$ and $t0 \leq t$
 shows $[\lambda s. \vartheta s < 0] \subseteq \text{wp} (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [\lambda s. \vartheta s < 0]$
 apply($\text{subgoal-tac } [\lambda s. \vartheta s < 0] \subseteq \text{wp} (\{[x'=f]\{t0..t\} S @ (Inf \{t0..t\}) \ \& \ G\}) \ [\lambda s. \vartheta s < 0]$)
 apply($\text{subgoal-tac Inf } \{t0..t\} = t0$, simp)
 using $\langle t0 \leq t \rangle$ apply($\text{simp add: closed-segment-eq-real-ivl}$)
 apply($\text{rule invariant-below-0[of - } \{t0..t\} \text{ - } \nu]$)
 using $\text{assms by(auto simp: closed-segment-eq-real-ivl)}$

theorem *dInvariant-below-0*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow$
 $(\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x r))) \text{ (at } r$
 $\text{within } \{t0..\tau\}))$
and $\text{impls}: [P] \subseteq [\lambda s. \vartheta s < 0] \ [\lambda s. \vartheta s < 0] \subseteq [Q] \ [G] \subseteq [\lambda s. \nu s \leq 0]$
and $t0 \leq t$
shows $[P] \subseteq \text{wp } (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [Q]$
using $\langle t0 \leq t \rangle$ **apply**(*rule-tac* $C = \lambda s. \vartheta s < 0$ **in** *dCut-interval*, *simp* *add*: $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $[\lambda s. \vartheta s < 0] \subseteq \text{wp } (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [\lambda s. \vartheta s < 0]$)
using *impls* **apply** *blast*
apply(*rule-tac* $\nu = \nu$ **in** *invariant-below-0-interval*)
using *assms*(1,4,5) **apply**(*simp*, *simp*, *simp*)
apply(*rule* *dWeakening*)
using *impls* **by** *simp*

lemma *invariant-meet*:

assumes $[I1] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [I1]$
and $[I2] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [I2]$
shows $[\lambda s. I1 s \wedge I2 s] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [\lambda s. I1 s \wedge I2 s]$
using *assms* **apply**(*subst* (*asm*) *wp-rel*, *subst* (*asm*) *wp-rel*)
apply(*subst* *wp-rel*, *simp* *add*: *p2r-def*)
by *blast*

theorem *dInvariant-meet*:

assumes $[I1] \subseteq \text{wp } (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [I1]$ **and** $[I2] \subseteq \text{wp}$
 $(\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [I2]$
and $\text{impls}: [P] \subseteq [\lambda s. I1 s \wedge I2 s] \ [\lambda s. I1 s \wedge I2 s] \subseteq [Q]$ **and** $t0 \leq t$
shows $[P] \subseteq \text{wp } (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [Q]$
apply(*rule-tac* $C = \lambda s. I1 s \wedge I2 s$ **in** *dCut-interval*, *simp* *add*: $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $[\lambda s. I1 s \wedge I2 s] \subseteq \text{wp } (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ [\lambda s.$
 $I1 s \wedge I2 s]$)
using *impls* **apply** *blast*
apply(*rule* *invariant-meet*)
using *assms*(1,2,5) **apply**(*simp*, *simp*)
apply(*rule* *dWeakening*)
using *impls* **by** *simp*

lemma *invariant-join*:

assumes $[I1] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [I1]$
and $[I2] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [I2]$
shows $[\lambda s. I1 s \vee I2 s] \subseteq \text{wp } (\{[x'=f] T S @ t0 \ \& \ G\}) \ [\lambda s. I1 s \vee I2 s]$
using *assms* **apply**(*subst* (*asm*) *wp-rel*, *subst* (*asm*) *wp-rel*)
apply(*subst* *wp-rel*, *simp* *add*: *p2r-def*)
by *blast*


```

theorem dInvariant-join:
  assumes  $\lceil I1 \rceil \subseteq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I1 \rceil$  and  $\lceil I2 \rceil \subseteq wp$ 
 $(\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I2 \rceil$ 
  and impls:  $\lceil P \rceil \subseteq \lceil \lambda s. I1 \ s \vee I2 \ s \rceil \ \lceil \lambda s. I1 \ s \vee I2 \ s \rceil \subseteq \lceil Q \rceil$  and  $t0 \leq t$ 
  shows  $\lceil P \rceil \subseteq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil Q \rceil$ 
  apply(rule-tac  $C=\lambda s. I1 \ s \vee I2 \ s$  in dCut-interval, simp add: t0 ≤ t)
  apply(subgoal-tac  $\lceil \lambda s. I1 \ s \vee I2 \ s \rceil \subseteq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil \lambda s.$ 
 $I1 \ s \vee I2 \ s \rceil$ )
  using impls apply blast
  apply(rule invariant-join)
  using assms(1,2,5) apply(simp, simp)
  apply(rule dWeakening)
  using impls by auto

end
theory cat2rel-examples
  imports cat2rel

begin

```

4.3 Examples

Here we do our first verification example: the single-evolution ball. We do it in two ways. The first one provides (1) a finite type and (2) its corresponding problem-specific vector-field and flow. The second approach uses an existing finite type and defines a more general vector-field which is later instantiated to the problem at hand.

4.3.1 Specific vector field

We define a finite type of three elements. All the lemmas below proven about this type must exist in order to do the verification example.

```

typedef three =  $\{m::nat. \ m < 3\}$ 
  apply(rule-tac  $x=0$  in exI)
  by simp

lemma CARD-of-three:  $CARD(three) = 3$ 
  using type-definition.card type-definition-three by fastforce

instance three::finite
  apply(standard, subst bij-betw-finite[of Rep-three UNIV {m::nat. m < 3}])
  apply(rule bij-betwI')
  apply (simp add: Rep-three-inject)
  using Rep-three apply blast
  apply (metis Abs-three-inverse UNIV-I)
  by simp

```

lemma *three-univD*: $(UNIV::three\ set) = \{Abs-three\ 0, Abs-three\ 1, Abs-three\ 2\}$
proof –
 have $(UNIV::three\ set) = Abs-three\ ` \{m::nat. m < 3\}$
 apply *auto by (metis Rep-three Rep-three-inverse image-iff)*
 also have $\{m::nat. m < 3\} = \{0, 1, 2\}$ **by** *auto*
 ultimately show *?thesis* **by** *auto*
qed

lemma *three-exhaust*: $\forall\ x::three. x = Abs-three\ 0 \vee x = Abs-three\ 1 \vee x = Abs-three\ 2$
using *three-univD* **by** *auto*

Next we use our recently created type to generate a 3-dimensional vector space. We then define the vector field and the flow for the single-evolution ball on this vector space. Then we follow the standard procedure to prove that they are in fact a Lipschitz vector-field and a its flow.

abbreviation *free-fall-kinematics* $(s::real^three) \equiv (\chi\ i. \text{if } i=(Abs-three\ 0) \text{ then } s\ \$\ (Abs-three\ 1) \text{ else } \text{if } i=(Abs-three\ 1) \text{ then } s\ \$\ (Abs-three\ 2) \text{ else } 0)$

abbreviation *free-fall-flow* $t\ s \equiv (\chi\ i. \text{if } i=(Abs-three\ 0) \text{ then } s\ \$\ (Abs-three\ 2) \cdot t^2/2 + s\ \$\ (Abs-three\ 1) \cdot t + s\ \$\ (Abs-three\ 0) \text{ else if } i=(Abs-three\ 1) \text{ then } s\ \$\ (Abs-three\ 2) \cdot t + s\ \$\ (Abs-three\ 1) \text{ else } s\ \$\ (Abs-three\ 2))$

lemma *bounded-linear-free-fall-kinematics*:*bounded-linear free-fall-kinematics*
apply *unfold-locales*
 apply(*simp-all add: plus-vec-def scaleR-vec-def ext norm-vec-def L2-set-def*)
 apply(*rule-tac x=1 in exI, clarsimp*)
 apply(*subst three-univD, subst three-univD*)
 by(*auto simp: Abs-three-inject*)

lemma *free-fall-kinematics-continuous-on*: *continuous-on X free-fall-kinematics*
using *bounded-linear-free-fall-kinematics linear-continuous-on* **by** *blast*

lemma *free-fall-kinematics-is-picard-ivp*: $0 \leq t \implies t < 1 \implies$
picard-ivp $(\lambda\ t\ s. \text{free-fall-kinematics } s) \{0..t\}\ UNIV\ 1\ 0$
 unfolding *picard-ivp-def picard-ivp-axioms-def ubc-definitions*
 apply(*simp-all add: nonempty-set-def lipschitz-on-def, safe*)
 apply(*rule continuous-on-compose2[of UNIV - {0..t} \times UNIV snd]*)
 apply(*simp-all add: free-fall-kinematics-continuous-on continuous-on-snd*)
 apply(*simp add: dist-vec-def L2-set-def dist-real-def*)
 apply(*subst three-univD, subst three-univD*)
 by(*simp add: Abs-three-inject*)

lemma *free-fall-flow-solves-free-fall-kinematics*:
 $((\lambda\ \tau. \text{free-fall-flow } \tau\ s) \text{ solves-ode } (\lambda\ t\ s. \text{free-fall-kinematics } s)) \{0..t\}\ UNIV$
apply (*rule solves-vec-lambda*)

```

apply(simp add: solves-ode-def)
unfolding has-vderiv-on-def has-vector-derivative-def apply(auto simp: Abs-three-inject)
using poly-derivatives(3, 4) unfolding has-vderiv-on-def has-vector-derivative-def
by auto

```

We end the first example by computing the wlp of the kinematics for the single-evolution ball and then using it to verify "its safety".

corollary *free-fall-flow-DS*:

```

assumes  $0 \leq t$  and  $t < 1$ 
shows wp  $\{[x' = \lambda t \text{ s. free-fall-kinematics } s] \{0..t\} \text{ UNIV } @ 0 \ \& \ G\} [Q] =$ 
 $\llbracket \lambda x. \forall \tau \in \{0..t\}. (\forall r \in \{0..-\tau\}. G \text{ (free-fall-flow } r \ x)) \longrightarrow Q \text{ (free-fall-flow } \tau \ x) \rrbracket$ 
apply(subst picard-ivp.wp-g-orbit[of  $\lambda t \text{ s. free-fall-kinematics } s - - 1 - (\lambda t \text{ x. free-fall-flow } t \ x)$ ])
using free-fall-kinematics-is-picard-ivp and assms apply blast apply(clarify,
rule conjI)
using free-fall-flow-solves-free-fall-kinematics apply blast
apply(simp add: vec-eq-iff) using three-exhaust by auto

```

lemma *single-evolution-ball*:

```

assumes  $0 \leq t$  and  $t < 1$ 
shows
 $\llbracket \lambda s. (0::\text{real}) \leq s \ \$ (Abs-three \ 0) \wedge s \ \$ (Abs-three \ 0) = H \wedge s \ \$ (Abs-three \ 1) =$ 
 $0 \wedge 0 > s \ \$ (Abs-three \ 2) \rrbracket$ 
 $\subseteq \text{wp } (\{[x' = \lambda t \text{ s. free-fall-kinematics } s] \{0..t\} \text{ UNIV } @ 0 \ \& \ (\lambda s. s \ \$ (Abs-three$ 
 $0) \geq 0)\})$ 
 $\llbracket \lambda s. 0 \leq s \ \$ (Abs-three \ 0) \wedge s \ \$ (Abs-three \ 0) \leq H \rrbracket$ 
apply(subst free-fall-flow-DS)
by(simp-all add: assms mult-nonneg-nonpos2)

```

4.3.2 General vector field

It turns out that there is already a 3-element type:

```

term  $x::3$ 
lemma  $CARD(three) = CARD(3)$ 
unfolding CARD-of-three by simp

```

In fact, for each natural number n there is already a corresponding n -element type in Isabelle. However, there are still some lemmas that one needs to prove in order to use it in verification in n -dimensional vector spaces.

lemma *exhaust-5*: — The analog for 3 has already been proven in Analysis.

```

fixes  $x::5$ 
shows  $x=1 \vee x=2 \vee x=3 \vee x=4 \vee x=5$ 
proof (induct  $x$ )
case (of-int  $z$ )
then have  $0 \leq z$  and  $z < 5$  by simp-all
then have  $z = 0 \vee z = 1 \vee z = 2 \vee z = 3 \vee z = 4$  by arith
then show ?case by auto

```

qed

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

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

Next, we prove that every linear system of differential equations (i.e. it can be rewritten as $x' = A \cdot x$) satisfies the conditions of the Picard-Lindelöf theorem:

lemma *matrix-lipschitz-constant*:
fixes *A*::*real*^(*n*::*finite*) ^*n*
shows $\text{dist } (A *v x) (A *v y) \leq (\text{real } \text{CARD}('n))^2 \cdot \text{maxAbs } A \cdot \text{dist } x \ y$
unfolding *dist-norm vector-norm-distr-minus* **proof**(*subst norm-matrix-sgn*)
have $\text{norm}_S A \leq \text{maxAbs } A \cdot (\text{real } \text{CARD}('n) \cdot \text{real } \text{CARD}('n))$
by (*metis* (*no-types*) *Groups.mult-ac(2)* *norms-le-dims-maxAbs*)
then have $\text{norm}_S A \cdot \text{norm } (x - y) \leq (\text{real } (\text{card } (\text{UNIV}::'n \text{ set})))^2 \cdot \text{maxAbs } A \cdot \text{norm } (x - y)$
by (*simp add: cross3-simps(11) mult-left-mono semiring-normalization-rules(29)*)
also have $\text{norm } (A *v \text{sgn } (x - y)) \cdot \text{norm } (x - y) \leq \text{norm}_S A \cdot \text{norm } (x - y)$
by (*simp add: norm-sgn-le-norms cross3-simps(11) mult-left-mono*)
ultimately show $\text{norm } (A *v \text{sgn } (x - y)) \cdot \text{norm } (x - y) \leq (\text{real } \text{CARD}('n))^2 \cdot \text{maxAbs } A \cdot \text{norm } (x - y)$
using *order-trans-rules(23)* **by** *blast*
qed

lemma *picard-ivp-linear-system*:
fixes *A*::*real*^(*n*::*finite*) ^*n*
assumes $0 < ((\text{real } \text{CARD}('n))^2 \cdot (\text{maxAbs } A))$ (**is** $0 < ?L$)
assumes $0 \leq t$ **and** $t < 1 / ?L$
shows *picard-ivp* ($\lambda t \ s. A *v s$) {0..*t*} *UNIV* ?*L* 0
apply *unfold-locales*
subgoal **by**(*simp*, *metis continuous-on-compose2 continuous-on-cong continuous-on-id continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest*)
subgoal **using** *matrix-lipschitz-constant maxAbs-ge-0 zero-compare-simps(4,12)*
unfolding *lipschitz-on-def* **by** *blast*
apply(*simp-all add: assms*)
subgoal **for** *r s* **apply**(*subgoal-tac* $|r - s| < 1 / ((\text{real } \text{CARD}('n))^2 \cdot \text{maxAbs } A)$)
apply(*subst (asm) pos-less-divide-eq[of ?L |r - s| 1]*)
using *assms* **by** *auto*
done

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

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

abbreviation *K* \equiv (χ i. if i= (0::3) then *axis* (1::3) (1::real) else if i= 1 then *axis* 2 1 else 0)

abbreviation *flow-for-K* t s \equiv (χ i. if i= (0::3) then s \$ 2 · t ^ 2/2 + s \$ 1 · t + s \$ 0
else if i=1 then s \$ 2 · t + s \$ 1 else s \$ 2)

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

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

lemma *K-is-picard-ivp*: $0 \leq t \implies t < 1/9 \implies$
picard-ivp (λ t s. *K* *v s) {0..t} UNIV ((real *CARD*(3))² · maxAbs *K*) 0
apply(rule *picard-ivp-linear-system*)
unfolding *entries-K* **by** *auto*

lemma *flow-for-K-solves-K*: ((λ τ . *flow-for-K* τ s) *solves-ode* (λ t s. *K* *v s))
{0..t} UNIV
apply (rule *solves-vec-lambda*)
apply(*simp* add: *solves-ode-def*)
using *poly-derivatives*(1, 3, 4)
by(*auto simp: matrix-vector-mult-def*)

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

corollary *flow-for-K-DS*:
assumes $0 \leq t$ **and** $t < 1/9$
shows *wp* {[$x' = \lambda t$ s. *K* *v s]{0..t} UNIV @ 0 & *G*} [*Q*] =
[λ x. $\forall \tau \in \{0..t\}. (\forall r \in \{0 \dots \tau\}. G (flow-for-K r x)) \longrightarrow Q (flow-for-K \tau x)$]
apply(subst *picard-ivp.wp-g-orbit*[of λt s. *K* *v s - - ((real *CARD*(3))² · maxAbs *K*)
- (λt x. *flow-for-K* t x)])
using *K-is-picard-ivp* **and** *assms* **apply** *blast* **apply**(*clarify*, rule *conjI*)
using *flow-for-K-solves-K* **apply** *blast*
apply(*simp* add: *vec-eq-iff*) **using** *exhaust-3* **apply** *force*
by *simp*

lemma *single-evolution-ball-K*:
assumes $0 \leq t$ **and** $t < 1/9$
shows [$\lambda s. (0::real) \leq s$ \$ (0::3) \wedge s \$ 0 = *H* \wedge s \$ 1 = 0 \wedge 0 > s \$ 2]
 \subseteq *wp* ([$x' = \lambda t$ s. *K* *v s]{0..t} UNIV @ 0 & ($\lambda s. s$ \$ 0 \geq 0)) [$\lambda s. 0 \leq s$ \$
0 \wedge s \$ 0 $\leq H$]

apply(*subst flow-for-K-DS*)
using *assms* **by**(*simp-all add: mult-nonneg-nonpos2*)

4.3.3 Bouncing Ball with solution

Armed now with two vector fields for free-fall kinematics and their respective flows, proving the safety of a “bouncing ball” is merely an exercise of real arithmetic:

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

lemma [*bb-real-arith*]: $0 \leq x \implies 0 > g \implies 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \implies (x::real) \leq H$

proof–

assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$

then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*

hence $*:v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$

using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)

from this have $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*

also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$

using *divide-nonneg-neg* **by** *fastforce*

ultimately have $H - x \geq 0$ **by** *linarith*

thus *?thesis* **by** *auto*

qed

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$

and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$

shows $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$

proof–

from pos have $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*

then have $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$

by (*metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right monoid-mult-class.power2-eq-square semiring-class.distrib-left*)

hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot H = 0$

using *invar* **by** (*simp add: monoid-mult-class.power2-eq-square*)

from this have $(g \cdot \tau + v)^2 + 2 \cdot g \cdot H = 0$

apply(*subst power2-sum*) **by** (*metis (no-types, hide-lams) Groups.add-ac(2, 3)*)

Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2))

hence $2 \cdot g \cdot H + (- ((g \cdot \tau) + v))^2 = 0$

by (*metis Groups.add-ac(2) power2-minus*)

thus *?thesis*

by (*simp add: monoid-mult-class.power2-eq-square*)

qed

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$

shows $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$

$2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs = ?rhs*)

proof–

```

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

```

lemma *bouncing-ball*:

```

assumes  $0 \leq t$  and  $t < 1/9$ 
shows  $\lceil \lambda s. (0::\text{real}) \leq s \ \$ (0::3) \wedge s \ \$ 0 = H \wedge s \ \$ 1 = 0 \wedge 0 > s \ \$ 2 \rceil \subseteq wp$ 
  (( $\{x' = \lambda t s. K * v \ s\} \{0..t\}$  UNIV @  $0 \ \& \ (\lambda s. s \ \$ 0 \geq 0)$ );
  (IF  $(\lambda s. s \ \$ 0 = 0)$  THEN  $([1 ::= (\lambda s. - s \ \$ 1)])$  ELSE Id FI))*
   $\lceil \lambda s. 0 \leq s \ \$ 0 \wedge s \ \$ 0 \leq H \rceil$ 
  apply(rule rel-ad-mka-starI [of -  $\lceil \lambda s. 0 \leq s \ \$ (0::3) \wedge 0 > s \ \$ 2 \wedge$ 
     $2 \cdot s \ \$ 2 \cdot s \ \$ 0 = 2 \cdot s \ \$ 2 \cdot H + (s \ \$ 1 \cdot s \ \$ 1) \rceil$ ])
    apply(simp, simp only: rel-antidomain-kleene-algebra.fbox-seq)
    apply(subst p2r-r2p-wp-sym[of (IF  $(\lambda s. s \ \$ 0 = 0)$  THEN  $([1 ::= (\lambda s. - s$ 
       $\ \$ 1)])$  ELSE Id FI)])
    apply(subst flow-for-K-DS) using assms apply(simp, simp) apply(subst wp-trafo)
    by(auto simp: p2r-def rel-antidomain-kleene-algebra.cond-def
      rel-antidomain-kleene-algebra.ads-d-def rel-ad-def closed-segment-eq-real-ivl
      bb-real-arith)

```

4.3.4 Bouncing Ball with invariants

lemma *gravity-is-invariant*: $(x \text{ solves-ode } (\lambda t. (*v) K)) \{0..t\} \text{ UNIV} \implies \tau \in \{0..t\} \implies r \in \{0..\tau\} \implies$
 $((\lambda \tau. x \ \tau \ \$ 2) \text{ has-derivative } (\lambda \tau. \tau *_{\mathbb{R}} 0)) \text{ (at } r \text{ within } \{0..\tau\})$
 apply(drule-tac $i=2$ in solves-vec-nth)
 apply(unfold solves-ode-def has-vderiv-on-def has-vector-derivative-def, clarify)
 apply(erule-tac $x=r$ in ballE, simp add: matrix-vector-mult-def)
 by (simp-all add: has-derivative-within-subset)

lemma *bouncing-ball-invariant*: $(x \text{ solves-ode } (\lambda t. (*v) K)) \{0..t\} \text{ UNIV} \implies \tau \in \{0..t\} \implies$
 $r \in \{0..\tau\} \implies ((\lambda \tau. 2 \cdot x \ \tau \ \$ 2 \cdot x \ \tau \ \$ 0 - 2 \cdot x \ \tau \ \$ 2 \cdot H - x \ \tau \ \$ 1 \cdot x \ \tau \ \$$
 $1) \text{ has-derivative } (\lambda \tau. \tau *_{\mathbb{R}} 0)) \text{ (at } r \text{ within } \{0..\tau\})$
 apply(frul-tac $i=2$ in solves-vec-nth, frul-tac $i=1$ in solves-vec-nth, drule-tac $i=0$ in solves-vec-nth)

```

apply(unfold solves-ode-def has-vderiv-on-def has-vector-derivative-def, clarify)
apply(erule-tac x=r in ballE, simp-all add: matrix-vector-mult-def)+
apply(rule-tac f'1=λt. 2 · x r $ 2 · (t · x r $ 1)
  and g'1=λt. 2 · (t · (x r $ 1 · x r $ 2)) in derivative-eq-intros(11))
apply(rule-tac f'1=λt. 2 · x r $ 2 · (t · x r $ 1) and g'1=λt. 0 in
derivative-eq-intros(11))
apply(rule-tac f'1=λt. 0 and g'1=(λxa. xa · x r $ 1) in derivative-eq-intros(12))
apply(rule-tac g'1=λt. 0 in derivative-eq-intros(6), simp-all add: has-derivative-within-subset)
apply(rule-tac g'1=λt. 0 in derivative-eq-intros(7))
apply(rule-tac g'1=λt. 0 in derivative-eq-intros(6), simp-all add: has-derivative-within-subset)
by(rule-tac f'1=(λxa. xa · x r $ 2) and g'1=(λxa. xa · x r $ 2) in derivative-eq-intros(12),

  simp-all add: has-derivative-within-subset)

```

lemma *bouncing-ball-invariants:*

```

assumes 0 ≤ t and t < 1/9
shows[λs. (0::real) ≤ s $ (0::3) ∧ s $ 0 = H ∧ s $ 1 = 0 ∧ 0 > s $ 2] ⊆ wp
  ({x'=λt s. K * v s}{0..t} UNIV @ 0 & (λ s. s $ 0 ≥ 0));
  (IF (λ s. s $ 0 = 0) THEN ([1 ::= (λs. - s $ 1)]) ELSE Id FI))*
  [λs. 0 ≤ s $ 0 ∧ s $ 0 ≤ H]
apply(rule-tac I=[λs. 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-sym[of (IF (λs. s $ 0 = 0) THEN ([1 ::= (λs. - s
$ 1)]) ELSE Id FI)])
using assms(1) apply(rule dCut-interval[of - - - - λ s. s $ 2 < 0])
apply(rule-tac ϑ=λs. s $ 2 and ν=λs. 0 in dInvariant-below-0)
using gravity-is-invariant apply force
apply(simp, simp, simp, simp add: ⟨0 ≤ t⟩)
apply(rule-tac C=λ s. 2 · s$2 · s$0 - 2 · s$2 · H - s$1 · s$1 = 0 in
dCut-interval, simp add: ⟨0 ≤ t⟩)
apply(rule-tac ϑ=λs. 2 · s$2 · s$0 - 2 · s$2 · H - s$1 · s$1 and ν=λ s. 0
in dInvariant-eq-0)
using bouncing-ball-invariant apply force
apply(simp, simp, simp, simp add: ⟨0 ≤ t⟩)
apply(rule dWeakening, subst p2r-r2p-wp)
by(auto simp: bb-real-arith p2r-def rel-antidomain-kleene-algebra.cond-def
  rel-antidomain-kleene-algebra.fbox-def rel-antidomain-kleene-algebra.ads-d-def
rel-ad-def)

```

4.3.5 Circular motion with invariants

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

lemma [*simp*]:*i ≠ (0::2) ⟹ i = 1 using exhaust-2 by fastforce*

lemma *UNIV-2:(UNIV::2 set) = {0, 1}*

apply *safe using exhaust-2 two-eq-zero by auto*

lemma *sum-axis-UNIV-2* [*simp*]: $(\sum j \in (UNIV::2 \text{ set}). \text{axis } i \text{ } r \text{ } \$ j \cdot f j) = r \cdot (f::2 \Rightarrow \text{real}) \text{ } i$
unfolding *axis-def UNIV-2* **by** *simp*

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

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

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

lemma *Circ-is-picard-ivp*: $0 \leq t \implies t < 1/4 \implies$
picard-ivp $(\lambda t \text{ } s. \text{ Circ } *v \text{ } s) \{0..t\} UNIV ((\text{real CARD}(2))^2 \cdot \text{maxAbs } \text{Circ}) \text{ } 0$
apply (*rule picard-ivp-linear-system*)
unfolding *entries-Circ* **by** *auto*

lemma *flow-for-Circ-solves-Circ*: $((\lambda \tau. \text{ flow-for-Circ } \tau \text{ } s) \text{ solves-ode } (\lambda t \text{ } s. \text{ Circ } *v \text{ } s)) \{0..t\} UNIV$
apply (*rule solves-vec-lambda, clarsimp*)
subgoal for *i* **apply** (*cases i=0*)
apply (*simp-all add: matrix-vector-mult-def*)
unfolding *solves-ode-def has-vderiv-on-def has-vector-derivative-def* **apply** *auto*
subgoal for *x*
apply (*rule-tac f'1=λt. - s\$0 · (t · sin x) and g'1=λt. s\$1 · (t · cos x) in derivative-eq-intros(11)*)
apply (*rule derivative-eq-intros(6)[of cos (λxa. - (xa · sin x))]*)
apply (*rule-tac Db1=1 in derivative-eq-intros(58)*)
apply (*rule ssubst[of (·) 1 id], force, simp, force, force*)
apply (*rule derivative-eq-intros(6)[of sin (λxa. (xa · cos x))]*)
apply (*rule-tac Db1=1 in derivative-eq-intros(55)*)
apply (*rule ssubst[of (·) 1 id], force, simp, force, force*)
by (*simp add: Groups.mult-ac(3) Rings.ring-distrib(4)*)
subgoal for *x*
apply (*rule-tac f'1=λt. s\$0 · (t · cos x) and g'1=λt. - s\$1 · (t · sin x) in derivative-eq-intros(8)*)
apply (*rule derivative-eq-intros(6)[of sin (λxa. xa · cos x)]*)
apply (*rule-tac Db1=1 in derivative-eq-intros(55)*)
apply (*rule ssubst[of (·) 1 id], force, simp, force, force*)
apply (*rule derivative-eq-intros(6)[of cos (λxa. - (xa · sin x))]*)
apply (*rule-tac Db1=1 in derivative-eq-intros(58)*)
apply (*rule ssubst[of (·) 1 id], force, simp, force, force*)
by (*simp add: Groups.mult-ac(3) Rings.ring-distrib(4)*)
done
done

corollary *flow-for-Circ-DS*:

assumes $0 \leq t$ **and** $t < 1/4$
shows $wp \{[x' = \lambda t s. \text{Circ} * v s] \{0..t\} \text{UNIV} @ 0 \ \& \ G\} \lceil Q \rceil =$
 $\lceil \lambda x. \forall \tau \in \{0..t\}. (\forall r \in \{0 - - \tau\}. G (\text{flow-for-Circ } r x)) \longrightarrow Q (\text{flow-for-Circ } \tau x) \rceil$
apply(*subst picard-ivp.wp-g-orbit*[*of* $\lambda t s. \text{Circ} * v s - - ((\text{real CARD}(2))^2 \cdot \text{max-Abs Circ}) - (\lambda t x. \text{flow-for-Circ } t x)$])
using *Circ-is-picard-ivp* **and** *assms* **apply** *blast* **apply**(*clarify*, *rule conjI*)
using *flow-for-Circ-solves-Circ* **apply** *blast*
apply(*simp add: vec-eq-iff*) **using** *exhaust-2 two-eq-zero* **apply** *force*
by *simp*

lemma *circular-motion*:

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

end

theory *cat2funcset-pre*

imports *../flow-locale Transformer-Semantics.Kleisli-Quantale KAD.Modal-Kleene-Algebra*

begin

5 Preliminaries

This file presents a miscellaneous collection of preliminary lemmas for verification of Hybrid Systems in Isabelle.

— We start by deleting some conflicting notation and introducing some new.

no-notation *Archimedean-Field.ceiling* ($\lceil \cdot \rceil$)
and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor \cdot \rfloor$)
and *Range-Semiring.antirange-semiring-class.ars-r* (r)
and *Isotone-Transformers.bqtran* ($\lfloor \cdot \rfloor$)

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

type-synonym $'a \text{ pred} = 'a \Rightarrow \text{bool}$

5.1 Nondeterministic Functions

lemma *Abs-nd-fun-inverse2*[*simp*]: $(f\bullet)\bullet = f$
by(*simp add: Abs-nd-fun-inverse*)

lemma *nd-fun-ext*: $(\bigwedge x. (f\bullet) x = (g\bullet) x) \Longrightarrow f = g$
apply(*subgoal-tac Rep-nd-fun f = Rep-nd-fun g*)
using *Rep-nd-fun-inject* **apply** *blast*
by(*rule ext, simp*)

```

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

lift-definition antidomain-op-nd-fun :: 'a nd-fun  $\Rightarrow$  'a nd-fun
  is  $\lambda f. (\lambda x. \text{if } ((f \bullet) x = \{\}) \text{ then } \{x\} \text{ else } \{\})^\bullet.$ 
lift-definition zero-nd-fun :: 'a nd-fun
  is  $\zeta^\bullet.$ 
lift-definition star-nd-fun :: 'a nd-fun  $\Rightarrow$  'a nd-fun
  is  $\lambda(f :: 'a \text{ nd-fun}). \text{qstar } f.$ 
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 nondeter-
ministic functions.

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

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

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

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

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

lemma nd-fun-ad-zero[nd-fun-aka]:  $\text{ad } (x :: 'a \text{ nd-fun}) \cdot x = 0$ 
  and nd-fun-ad[nd-fun-aka]:  $\text{ad } (x \cdot y) + \text{ad } (x \cdot \text{ad } (\text{ad } y)) = \text{ad } (x \cdot \text{ad } (\text{ad } y))$ 
  and nd-fun-ad-one[nd-fun-aka]:  $\text{ad } (\text{ad } x) + \text{ad } x = 1$ 
  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]:  $1 + (x :: 'a \text{ nd-fun}) \cdot x^\star \leq x^\star$ 
  and nd-star-unfoldl[nd-fun-aka]:  $z + x \cdot y \leq y \implies x^\star \cdot z \leq y$ 
  and nd-star-unfoldr[nd-fun-aka]:  $z + y \cdot x \leq y \implies z \cdot x^\star \leq y$ 
  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

5.2 Weakest Liberal Preconditions

abbreviation *p2ndf* :: 'a pred \Rightarrow 'a nd-fun ((1[-]))
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:* $(\lceil P \rceil = \lceil Q \rceil) = (P = Q)$
proof(*safe*)
assume $\lceil P \rceil = \lceil Q \rceil$
hence $\lceil P \rceil \leq \lceil Q \rceil \wedge \lceil Q \rceil \leq \lceil P \rceil$ **by** *simp*
then have $(\forall s. P s \longrightarrow Q s) \wedge (\forall s. Q s \longrightarrow P s)$ **by** *simp*
thus $P = Q$ **by** *auto*
qed

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

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

lemma *p2ndf-ndf2p-id:* $F \leq \eta^\bullet \Longrightarrow \lfloor \lceil F \rceil \rfloor = F$
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*)

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

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

lemma *wp-nd-fun-etaD:* $\text{wp } (F_\bullet) \lceil P \rceil = \eta^\bullet \Longrightarrow (\forall y. y \in (F x) \longrightarrow P y)$
proof
fix *y* **assume** $\text{wp } (F_\bullet) \lceil P \rceil = (\eta^\bullet)$
from this have $\eta^\bullet = \lceil \lambda s. \forall y. s2p (F s) y \longrightarrow P y \rceil$
by(*subst wp-nd-fun[THEN sym], simp*)
hence $\bigwedge x. \{x\} = \{s. s = x \wedge (\forall y. s2p (F s) y \longrightarrow P y)\}$

apply(subst (asm) Abs-nd-fun-inject, simp-all)
by(drule-tac x=x in fun-cong, simp)
then show s2p (F x) y \longrightarrow P y **by** auto
qed

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

lemma p2ndf-ndf2p-wp-sym: $wp\ R\ P = \llbracket wp\ R\ P \rrbracket$
by(rule sym, simp add: p2ndf-ndf2p-wp)

lemma wp-trafo: $\llbracket wp\ F\ \llbracket Q \rrbracket \rrbracket = (\lambda s. \forall s'. s' \in (F\bullet) \ s \longrightarrow Q\ s')$
apply(subgoal-tac F = (F \bullet) \bullet)
apply(rule ssubst[of F (F \bullet) \bullet], simp)
apply(subst wp-nd-fun)
by(simp-all add: f2r-def)

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

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

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

lemma fbox-seq [simp]: $|x \cdot y|\ q = |x|\ |y|\ q$
by (simp add: fbox-mult)

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

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**
 $IF\ P\ THEN\ X\ ELSE\ Y\ FI \equiv cond\ \llbracket P \rrbracket\ X\ Y$

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 $d\ p \leq |x^*|\ q$
by (meson assms local.dual-order.trans local.fbox-iso local.fbox-star-induct-var)

lemma bot-pres-del:bot-pres $(If\ (\neg\ Q\ x)\ (\eta\ x)) \Longrightarrow Q\ x$
using empty-not-insert **by** fastforce **thm** empty-not-insert

lemma nd-fun-ads-d-def: $d\ (f::'a\ nd-fun) = (\lambda x. if\ (f\bullet)\ x = \{\}\ then\ \{\}\ else\ \eta\ x)^\bullet$

```

unfolding ads-d-def apply(rule nd-fun-ext, simp)
apply transfer by auto

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

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

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

```

5.3 Real Numbers and Derivatives

```

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

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

lemma sqrt-le-itself:  $1 \leq x \implies \text{sqrt}\ x \leq x$ 
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:  $\text{sqrt}\ (\text{real}\ n) \leq \text{real}\ n$ 
by (metis (full-types) abs-of-nat le-square of-nat-mono of-nat-mult real-sqrt-abs2
  real-sqrt-le-iff)

lemma semiring-factor-left:  $a \cdot b + a \cdot c = a \cdot ((b::('a::semiring)) + c)$ 
by(subst Groups.algebra-simps(18), simp)

```

lemma *sin-cos-squared-add3*: $(x::('a::\{\text{banach}, \text{real-normed-field}\})) \cdot (\sin t)^2 + x \cdot (\cos t)^2 = x$

by(*subst semiring-factor-left, subst sin-cos-squared-add, simp*)

lemma *sin-cos-squared-add4*: $(x::('a::\{\text{banach}, \text{real-normed-field}\})) \cdot (\cos t)^2 + x \cdot (\sin t)^2 = x$

by(*subst semiring-factor-left, subst sin-cos-squared-add2, simp*)

lemma [*simp*]: $((x::\text{real}) \cdot \cos t - y \cdot \sin t)^2 + (x \cdot \sin t + y \cdot \cos t)^2 = x^2 + y^2$

proof–

have $(x \cdot \cos t - y \cdot \sin t)^2 = x^2 \cdot (\cos t)^2 + y^2 \cdot (\sin t)^2 - 2 \cdot (x \cdot \cos t) \cdot (y \cdot \sin t)$

by(*simp add: power2-diff power-mult-distrib*)

also have $(x \cdot \sin t + y \cdot \cos t)^2 = y^2 \cdot (\cos t)^2 + x^2 \cdot (\sin t)^2 + 2 \cdot (x \cdot \cos t) \cdot (y \cdot \sin t)$

by(*simp add: power2-sum power-mult-distrib*)

ultimately show $(x \cdot \cos t - y \cdot \sin t)^2 + (x \cdot \sin t + y \cdot \cos t)^2 = x^2 + y^2$

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

qed

lemma *closed-segment-mvt*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes $(\bigwedge r. r \in \{a \dots b\} \implies (f \text{ has-derivative } f' r) \text{ (at } r \text{ within } \{a \dots b\}))$ **and**
 $a \leq b$

shows $\exists r \in \{a \dots b\}. f b - f a = f' r (b - a)$

using *assms closed-segment-eq-real-ivl and mvt-very-simple by auto*

lemma *convergences-solves-vec-nth*:

assumes $((\lambda y. (\varphi y - \varphi (\text{netlimit (at } x \text{ within } \{0..t\}))) - (y - \text{netlimit (at } x \text{ within } \{0..t\}))) *_R f (\varphi x)) /_R$

$|y - \text{netlimit (at } x \text{ within } \{0..t\})| \longrightarrow 0) \text{ (at } x \text{ within } \{0..t\})$ **(is** $((\lambda y. ?f y) \longrightarrow 0) ?net)$

shows $((\lambda y. (\varphi y \$ i - \varphi (\text{netlimit (at } x \text{ within } \{0..t\}))) \$ i - (y - \text{netlimit (at } x \text{ within } \{0..t\}))) *_R f (\varphi x) \$ i) /_R$

$|y - \text{netlimit (at } x \text{ within } \{0..t\})| \longrightarrow 0) \text{ (at } x \text{ within } \{0..t\})$ **(is** $((\lambda y. ?g y i) \longrightarrow 0) ?net)$

proof–

from *assms* **have** $((\lambda y. ?f y \$ i) \longrightarrow 0 \$ i) ?net$ **by**(*rule tendsto-vec-nth*)

also have $(\lambda y. ?f y \$ i) = (\lambda y. ?g y i)$ **by** *auto*

ultimately show $((\lambda y. ?g y i) \longrightarrow 0) ?net$ **by** *auto*

qed

lemma *solves-vec-nth*:

fixes $f::('a::\text{banach})^{('n::\text{finite})} \Rightarrow ('a^{^n})$

assumes $(\varphi \text{ solves-ode } (\lambda t. f)) \{0..t\}$ *UNIV*

shows $((\lambda t. (\varphi t) \$ i) \text{ solves-ode } (\lambda t s. (f (\varphi t)) \$ i)) \{0..t\}$ *UNIV*

using *assms unfolding solves-ode-def has-vderiv-on-def has-vector-derivative-def has-derivative-def*

```

apply safe apply(auto simp: bounded-linear-def bounded-linear-axioms-def)[1]
apply(erule-tac x=x in ballE, clarsimp)
apply(rule convergences-solves-vec-nth)
by(simp-all add: Pi-def)

lemma solves-vec-lambda:
  fixes f::('a::banach) ^ ('n::finite)  $\Rightarrow$  ('a ^ 'n) and  $\varphi$ ::real  $\Rightarrow$  ('a ^ 'n)
  assumes  $\forall i::'n. ((\lambda t. (\varphi t) \$ i) \text{ solves-ode } (\lambda t s. (f (\varphi t)) \$ i)) \{0..t\} \text{ UNIV}$ 
  shows  $(\varphi \text{ solves-ode } (\lambda t. f)) \{0..t\} \text{ UNIV}$ 
  using assms unfolding solves-ode-def has-vderiv-on-def has-vector-derivative-def
  has-derivative-def
  apply safe apply(auto simp: bounded-linear-def bounded-linear-axioms-def)[1]
  by(rule Finite-Cartesian-Product.vec-tendstoI, simp-all)

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

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

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

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

lemma quadratic-monomial-derivative:
  (( $\lambda t::\text{real}. a \cdot t^2$ ) 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-derivative-div:
  (( $\lambda t::\text{real}. a \cdot t^2 / 2$ ) 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-derivatives]:(( $\lambda t. a \cdot t^2 / 2$ ) has-vderiv-on
( $\cdot$ ) a) T
apply(simp add: has-vderiv-on-def has-vector-derivative-def, clarify)
using quadratic-monomial-derivative-div by (simp add: mult-commute-abs)

lemma pos-vderiv[poly-derivatives]:
  (( $\lambda t. a \cdot t^2 / 2 + v \cdot t + x$ ) 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(190))
  apply(rule-tac f'1= $\lambda x. a \cdot x$  and g'1= $\lambda x. v$  in derivative-intros(190))
  using poly-derivatives(2) by(auto intro: derivative-intros)

lemma pos-derivative:
   $t \in T \implies ((\lambda \tau. a \cdot \tau^2 / 2 + v \cdot \tau + x) \text{ has-derivative } (\lambda x. x *_R (a \cdot t + v)))$ 

```


(at t within T)
using *pos-vderiv* **unfolding** *has-vderiv-on-def* *has-vector-derivative-def* **by** *simp*

lemma *vel-vderiv*[*poly-derivatives*]: $((\lambda r. a \cdot r + v) \text{ has-vderiv-on } (\lambda t. a)) \ T$
apply(*rule-tac* $f'1 = \lambda x. a$ **and** $g'1 = \lambda x. 0$ **in** *derivative-intros*(190))
unfolding *has-vderiv-on-def* **by**(*auto* *intro*: *derivative-eq-intros*)

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

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

declare *origin-line-vector-derivative* [*poly-derivatives*]
and *origin-line-derivative* [*poly-derivatives*]
and *quadratic-monomial-derivative* [*poly-derivatives*]
and *quadratic-monomial-derivative-div* [*poly-derivatives*]
and *pos-derivative* [*poly-derivatives*]

5.4 Vectors, norms and matrices.

5.4.1 Unit vectors and vector norm

lemma *norm-scalar-mult*: $\text{norm } ((c::\text{real}) * s \ x) = |c| \cdot \text{norm } x$
unfolding *norm-vec-def* *L2-set-def* *real-norm-def* *vector-scalar-mult-def* **apply**
simp
apply(*subgoal-tac* $(\sum i \in \text{UNIV}. (c \cdot x \ \$ \ i)^2) = |c|^2 \cdot (\sum i \in \text{UNIV}. (x \ \$ \ i)^2)$)
apply(*simp* *add*: *real-sqrt-mult*)
apply(*simp* *add*: *sum-distrib-left*)
by (*meson* *power-mult-distrib*)

lemma *squared-norm-vec*: $(\text{norm } x)^2 = (\sum i \in \text{UNIV}. (x \ \$ \ i)^2)$
unfolding *norm-vec-def* *L2-set-def* **by** (*simp* *add*: *sum-nonneg*)

lemma *sgn-is-unit-vec*: $\text{sgn } x = 1 / \text{norm } x * s \ x$
unfolding *sgn-vec-def* *scaleR-vec-def* **by**(*simp* *add*: *vector-scalar-mult-def* *divide-inverse-commute*)

lemma *norm-sgn-unit*: $(x::\text{real}^n) \neq 0 \implies \text{norm } (\text{sgn } x) = 1$
proof(*subst* *sgn-is-unit-vec*, *unfold* *norm-vec-def* *L2-set-def*, *simp* *add*: *power-divide*)
assume $x \neq 0$
have $(\sum i \in \text{UNIV}. (x \ \$ \ i)^2 / (\text{norm } x)^2) = 1 / (\text{norm } x)^2 \cdot (\sum i \in \text{UNIV}. (x \ \$ \ i)^2)$
by (*simp* *add*: *sum-divide-distrib*)
also have $(\sum i \in \text{UNIV}. (x \ \$ \ i)^2) = (\text{norm } x)^2$ **by**(*subst* *squared-norm-vec*, *simp*)
ultimately show $(\sum i \in \text{UNIV}. (x \ \$ \ i)^2 / (\text{sqrt } (\sum i \in \text{UNIV}. (x \ \$ \ i)^2))^2) = 1$

using $\langle x \neq 0 \rangle$ by simp
qed

lemma norm-matrix-sgn: norm $(A * v (x :: \text{real}^n)) = \text{norm } (A * v (\text{sgn } x)) \cdot \text{norm } x$
unfolding sgn-is-unit-vec vector-scalar-commute norm-scalar-mult by simp

lemma vector-norm-distr-minus:
fixes $A :: ('a :: \{\text{real-normed-vector}, \text{ring-1}\})^n \times^m$
shows $\text{norm } (A * v x - A * v y) = \text{norm } (A * v (x - y))$
by (subst matrix-vector-mult-diff-distrib, simp)

5.4.2 Matrix norm

abbreviation norm_S $(A :: \text{real}^n \times^m) \equiv \text{Sup } \{\text{norm } (A * v x) \mid x. \text{norm } x = 1\}$

lemma unit-norms-bound:
fixes $A :: \text{real}^{n \times m}$
shows $\text{norm } x = 1 \implies \text{norm } (A * v x) \leq \text{norm } ((\chi \ i \ j. |A \$ i \$ j|) * v 1)$
proof –
assume $\text{norm } x = 1$
from this **have** $\bigwedge j. |x \$ j| \leq 1$
by (metis component-le-norm-cart)
then **have** $\bigwedge i \ j. |A \$ i \$ j| \cdot |x \$ j| \leq |A \$ i \$ j| \cdot 1$
using mult-left-mono **by** (simp add: mult-left-le)
from this **have** $\bigwedge i. (\sum j \in \text{UNIV}. |A \$ i \$ j| \cdot |x \$ j|)^2 \leq (\sum j \in \text{UNIV}. |A \$ i \$ j|)^2$
by (simp add: power-mono sum-mono sum-nonneg)
also **have** $\bigwedge i. (\sum j \in \text{UNIV}. A \$ i \$ j \cdot x \$ j)^2 \leq (\sum j \in \text{UNIV}. |A \$ i \$ j \cdot x \$ j|)^2$
using abs-le-square-iff **by** force
moreover **have** $\bigwedge i. (\sum j \in \text{UNIV}. |A \$ i \$ j \cdot x \$ j|)^2 = (\sum j \in \text{UNIV}. |A \$ i \$ j| \cdot |x \$ j|)^2$
by (simp add: abs-mult)
ultimately **have** $\bigwedge i. (\sum j \in \text{UNIV}. A \$ i \$ j \cdot x \$ j)^2 \leq (\sum j \in \text{UNIV}. |A \$ i \$ j| \cdot |x \$ j|)^2$
using order-trans **by** fastforce
hence $(\sum i \in \text{UNIV}. (\sum j \in \text{UNIV}. A \$ i \$ j \cdot x \$ j)^2) \leq (\sum i \in \text{UNIV}. (\sum j \in \text{UNIV}. |A \$ i \$ j| \cdot |x \$ j|)^2)$
by (simp add: sum-mono)
then **have** $(\text{sqrt } (\sum i \in \text{UNIV}. (\sum j \in \text{UNIV}. A \$ i \$ j \cdot x \$ j)^2)) \leq (\text{sqrt } (\sum i \in \text{UNIV}. (\sum j \in \text{UNIV}. |A \$ i \$ j| \cdot |x \$ j|)^2))$
using real-sqrt-le-mono **by** blast
thus $\text{norm } (A * v x) \leq \text{norm } ((\chi \ i \ j. |A \$ i \$ j|) * v 1)$
by (simp add: norm-vec-def L2-set-def matrix-vector-mult-def)
qed

lemma unit-norms-exists:
fixes $A :: \text{real}^{n \times m}$

```

shows bounded:bounded {norm (A * v x) | x. norm x = 1}
and bdd-above:bdd-above {norm (A * v x) | x. norm x = 1}
and non-empty:{norm (A * v x) | x. norm x = 1} ≠ {} (is ?U ≠ {})
proof –
  show bounded ?U
    apply(unfold bounded-def, rule-tac x=0 in exI, simp add: dist-real-def)
    apply(rule-tac x=norm ((χ i j. |A $ i $ j|) * v 1) in exI, clarsimp)
    using unit-norms-bound by blast
next
  show bdd-above ?U
    apply(unfold bdd-above-def, rule-tac x=norm ((χ i j. |A $ i $ j|) * v 1) in exI,
clarsimp)
    using unit-norms-bound by blast
next
  have  $\bigwedge k::'n. \text{norm } (\text{axis } k \ (1::\text{real})) = 1$ 
    using norm-axis-1 by blast
  hence  $\bigwedge k::'n. \text{norm } ((A::\text{real}^{('n::\text{finite})})^{('m)}) * v (\text{axis } k \ (1::\text{real}))) \in ?U$ 
    by blast
  thus ?U ≠ {} by blast
qed

lemma unit-norms:  $\text{norm } x = 1 \implies \text{norm } (A * v x) \leq \text{norm}_S A$ 
  using cSup-upper mem-Collect-eq unit-norms-exists(2) by (metis (mono-tags,
lifting))

lemma unit-norms-ge-0:  $0 \leq \text{norm}_S A$ 
  using ex-norm-eq-1 norm-ge-zero unit-norms basic-trans-rules(23) by blast

lemma norm-sgn-le-norms:  $\text{norm } (A * v \text{sgn } x) \leq \text{norm}_S A$ 
  apply(cases x=0)
  using sgn-zero unit-norms-ge-0 apply force
  using norm-sgn-unit unit-norms by blast

abbreviation entries (A::real^{('n^{('m)})})  $\equiv \{A \$ i \$ j \mid i j. i \in (\text{UNIV}::'m \text{ set}) \wedge j \in (\text{UNIV}::'n \text{ set})\}$ 
abbreviation maxAbs (A::real^{('n^{('m)})})  $\equiv \text{Max } (\text{abs } ' (\text{entries } A))$ 

lemma maxAbs-def:  $\text{maxAbs } (A::\text{real}^{('n^{('m)})}) = \text{Max } \{ |A \$ i \$ j| \mid i j. i \in (\text{UNIV}::'m \text{ set}) \wedge j \in (\text{UNIV}::'n \text{ set}) \}$ 
  apply(simp add: image-def, rule arg-cong[of - - Max])
  by auto

lemma finite-matrix-abs:
  fixes A::real^{('n::finite)^{('m::finite)}}
  shows finite { $|A \$ i \$ j| \mid i j. i \in (\text{UNIV}::'m \text{ set}) \wedge j \in (\text{UNIV}::'n \text{ set})$ } (is
finite ?X)
proof –
  {fix i::'m
    have finite { $|A \$ i \$ j| \mid j. j \in (\text{UNIV}::'n \text{ set})$ }}
```

using *finite-Atleast-Atmost-nat* by *fastforce*
 hence $\forall i::'m. \text{finite } \{|A \$ i \$ j| \mid j. j \in (UNIV::'n \text{ set})\}$ by *blast*
 then have *finite* $(\bigcup i \in UNIV. \{|A \$ i \$ j| \mid j. j \in (UNIV::'n \text{ set})\})$ (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 *maxAbs-ge-0*: $\text{maxAbs } A \geq 0$

proof—

have $\bigwedge i j. |A \$ i \$ j| \geq 0$ by *simp*
 also have $\bigwedge i j. \text{maxAbs } A \geq |A \$ i \$ j|$
 unfolding *maxAbs-def* using *finite-matrix-abs* *Max-ge* *maxAbs-def* by *blast*
 finally show $0 \leq \text{maxAbs } A$.
 qed

lemma *norms-le-dims-maxAbs*:

fixes $A::\text{real}^{('n::\text{finite}) \times ('m::\text{finite})}$
 shows $\text{norm}_S A \leq \text{real CARD}('n) \cdot \text{real CARD}('m) \cdot (\text{maxAbs } A)$ (is $\text{norm}_S A \leq ?n \cdot ?m \cdot (\text{maxAbs } A)$)

proof—

{fix $x::(\text{real}, 'n) \text{ vec}$ assume $\text{norm } x = 1$
 hence *comp-le-1*: $\forall i::'n. |x \$ i| \leq 1$
 by (*simp add: norm-bound-component-le-cart*)
 have $A * v x = (\sum i \in UNIV. x \$ i * s \text{ column } i A)$
 using *matrix-mult-sum* by *blast*
 hence $\text{norm } (A * v x) \leq (\sum (i::'n) \in UNIV. \text{norm } (x \$ i * s \text{ column } i A))$
 by (*simp add: sum-norm-le*)
 also have $\dots = (\sum (i::'n) \in UNIV. |x \$ i| \cdot \text{norm } (\text{column } i A))$
 by (*simp add: norm-scalar-mult*)
 also have $\dots \leq (\sum (i::'n) \in UNIV. \text{norm } (\text{column } i A))$
 by (*metis (no-types, lifting) Groups.mult-ac(2) comp-le-1 mult-left-le norm-ge-zero sum-mono*)

also have $\dots \leq (\sum (i::'n) \in UNIV. ?m \cdot \text{maxAbs } A)$
 proof(*unfold norm-vec-def L2-set-def real-norm-def*)
 have $\bigwedge i j. |\text{column } i A \$ j| \leq \text{maxAbs } A$
 using *finite-matrix-abs* *Max-ge* unfolding *column-def* *maxAbs-def* by(*simp, blast*)
 hence $\bigwedge i j. |\text{column } i A \$ j|^2 \leq (\text{maxAbs } A)^2$
 by (*metis (no-types, lifting) One-nat-def abs-ge-zero numerals(2) order-trans-rules(23)*)

power2-abs power2-le-iff-abs-le
 then have $\bigwedge i. (\sum j \in UNIV. |\text{column } i A \$ j|^2) \leq (\sum (j::'m) \in UNIV. (\text{maxAbs } A)^2)$
 by (*meson sum-mono*)
 also have $(\sum (j::'m) \in UNIV. (\text{maxAbs } A)^2) = ?m \cdot (\text{maxAbs } A)^2$ by *simp*
 ultimately have $\bigwedge i. (\sum j \in UNIV. |\text{column } i A \$ j|^2) \leq ?m \cdot (\text{maxAbs } A)^2$
 by *force*

```

    hence  $\bigwedge i. \text{sqrt} (\sum_{j \in \text{UNIV}. |\text{column } i \text{ } A \text{ } j|^2}) \leq \text{sqrt} (?m \cdot (\text{maxAbs } A)^2)$ 
    by (simp add: real-sqrt-le-mono)
    also have  $\text{sqrt} (?m \cdot (\text{maxAbs } A)^2) \leq \text{sqrt } ?m \cdot \text{maxAbs } A$ 
    using maxAbs-ge-0 real-sqrt-mult by auto
    also have  $\dots \leq ?m \cdot \text{maxAbs } A$ 
    using sqrt-real-nat-le maxAbs-ge-0 mult-right-mono by blast
    finally show  $(\sum_{i \in \text{UNIV}. \text{sqrt} (\sum_{j \in \text{UNIV}. |\text{column } i \text{ } A \text{ } j|^2})) \leq (\sum_{(i::'n) \in \text{UNIV}. ?m \cdot \text{maxAbs } A})$ 
    by (meson sum-mono)
  qed
  also have  $(\sum_{(i::'n) \in \text{UNIV}. (\text{maxAbs } A)) = ?n \cdot (\text{maxAbs } A)$ 
  using sum-constant-scale by auto
  ultimately have  $\text{norm} (A * v \ x) \leq ?n \cdot ?m \cdot (\text{maxAbs } A)$  by simp
  from this show ?thesis
  using unit-norms-exists[of A] Connected.bounded-has-Sup(2) by blast
qed

end
theory cat2funcset
  imports cat2funcset-pre

begin

```

6 Hybrid System Verification

6.1 Verification by providing solutions

```

abbreviation orbital f T S t0 x0  $\equiv$ 
   $\{x \ t \mid t \ x. \ t \in T \wedge (x \text{ solves-ode } f) T \ S \wedge x \ t0 = x0 \wedge x0 \in S \wedge t0 \in T\}$ 
abbreviation g-orbital f T S t0 x0 G  $\equiv$ 
   $\{x \ t \mid t \ x. \ t \in T \wedge (x \text{ solves-ode } f) T \ S \wedge x \ t0 = x0 \wedge x0 \in S \wedge t0 \in T \wedge (\forall \ r \in \{t0 \dots t\}. G \ (x \ r))\}$ 

abbreviation
  g-evolution  $:: (\text{real} \Rightarrow ('a::\text{banach}) \Rightarrow 'a) \Rightarrow \text{real set} \Rightarrow 'a \text{ set} \Rightarrow \text{real} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ nd-fun}$ 
   $((1 \{[x' = -] \text{ - } @ \text{ - } \& \text{ - } \}) \text{ where } \{[x' = f] T \ S @ t0 \ \& \ G\} \equiv (\lambda \ s. \text{g-orbital } f \ T \ S \ t0 \ s \ G)^\bullet$ 

context picard-ivp
begin

```

```

lemma orbital-collapses:
  assumes ivp:  $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } f) T \ S \wedge \varphi \ t0 \ s = s$  and  $s \in S$ 
  shows orbital f T S t0 s =  $\{\varphi \ t \ s \mid t. \ t \in T\}$ 
  apply safe apply (rule-tac x=t in exI, simp)
  apply (rule-tac x=xa and s=xa t0 in unique-solution, simp-all add: assms)
  apply (rule-tac x=t in exI, rule-tac x= $\lambda t. \varphi \ t \ s$  in exI)
  using assms init-time by auto

```

lemma *g-orbital-collapses*:

assumes *ivp*: $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } f) \ T \ S \wedge \varphi \ t0 \ s = s$ **and** $s \in S$
shows *g-orbital* $f \ T \ S \ t0 \ s \ G = \{\varphi \ t \ s \mid t. t \in T \wedge (\forall r \in \{t0 \dots t\}. G \ (\varphi \ r \ s))\}$
apply *safe* **apply**(*rule-tac* $x=t$ **in** *exI*, *simp*)
using *assms* *unique-solution* **apply**(*metis* *closed-segment-subset-domainI*)
apply(*rule-tac* $x=t$ **in** *exI*, *rule-tac* $x=\lambda t. \varphi \ t \ s$ **in** *exI*)
using *assms* *init-time* **by** *auto*

lemma *wp-orbit*:

assumes *ivp*: $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } f) \ T \ S \wedge \varphi \ t0 \ s = s$
shows *wp* $((\lambda s. \text{orbital } f \ T \ S \ t0 \ s)^\bullet) \ \lceil Q \rceil = \lceil \lambda s. \forall t \in T. s \in S \longrightarrow Q \ (\varphi \ t \ s) \rceil$
apply(*subst* *wp-nd-fun*, *subst* *eq-p2ndf-iff*) **apply**(*rule* *ext*, *safe*)
apply(*erule-tac* $x=\varphi \ t \ s$ **in** *allE*, *erule* *impE*, *simp*)
apply(*rule-tac* $x=t$ **in** *exI*, *rule-tac* $x=\lambda t. \varphi \ t \ s$ **in** *exI*)
using *ivp* *init-time* **apply**(*simp*, *simp*)
apply(*subgoal-tac* $\varphi \ t \ (x \ t0) = x \ t$)
apply(*erule-tac* $x=t$ **in** *ballE*, *simp*, *simp*)
by(*rule-tac* $y=x$ **and** $s=x \ t0$ **in** *unique-solution*, *simp-all* *add: assms*)

lemma *wp-g-orbit*:

assumes *ivp*: $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } f) \ T \ S \wedge \varphi \ t0 \ s = s$
shows *wp* $\{[x'=f] \ T \ S \ @ \ t0 \ \& \ G\} \ \lceil Q \rceil = \lceil \lambda s. \forall t \in T. s \in S \longrightarrow (\forall r \in \{t0 \dots t\}. G \ (\varphi \ r \ s)) \longrightarrow Q \ (\varphi \ t \ s) \rceil$
apply(*subst* *wp-nd-fun*, *subst* *eq-p2ndf-iff*) **apply**(*rule* *ext*, *safe*)
apply(*erule-tac* $x=\varphi \ t \ s$ **in** *allE*, *erule* *impE*, *simp*)
apply(*rule-tac* $x=t$ **in** *exI*, *rule-tac* $x=\lambda t. \varphi \ t \ s$ **in** *exI*)
apply(*simp* *add: ivp init-time*, *simp*)
apply(*subgoal-tac* $\forall r \in \{t0 \dots t\}. \varphi \ r \ (x \ t0) = x \ r$)
apply(*erule-tac* $x=t$ **in** *ballE*, *safe*)
apply(*erule-tac* $x=r$ **in** *ballE*) **+** **apply** *simp-all*
apply(*erule-tac* $x=t$ **in** *ballE*) **+** **apply** *simp-all*
apply(*rule-tac* $y=x$ **and** $s=x \ t0$ **in** *unique-solution*, *simp-all* *add: assms*)
using *subsegment* **by** *blast*

end

lemma *dSolution*:

assumes *picard-ivp* $f \ T \ S \ L \ t0$ **and** *ivp*: $\forall s \in S. ((\lambda t. \varphi \ t \ s) \text{ solves-ode } f) \ T \ S \wedge \varphi \ t0 \ s = s$
and $\forall s. P \ s \longrightarrow (\forall t \in T. s \in S \longrightarrow (\forall r \in \{t0..t\}. G \ (\varphi \ r \ s)) \longrightarrow Q \ (\varphi \ t \ s))$
shows $\lceil P \rceil \leq wp \ (\{[x'=f] \ T \ S \ @ \ t0 \ \& \ G\}) \ \lceil Q \rceil$
using *assms* **apply**(*subst* *picard-ivp.wp-g-orbit*, *auto*)
by (*simp* *add: Starlike.closed-segment-eq-real-ivl*)

This last theorem allows us to compute weakest liberal preconditions for known systems of ODEs:

corollary *line-DS*: $0 \leq t \implies wp \{[x'=\lambda t \ s. \ c]\{0..t\} \ UNIV \ @ \ 0 \ \& \ G\} \lceil Q \rceil =$
 $\lceil \lambda \ x. \ \forall \ \tau \in \{0..t\}. \ (\forall \ r \in \{0..-\tau\}. \ G \ (x + r *_{\mathcal{R}} c)) \longrightarrow Q \ (x + \tau *_{\mathcal{R}} c) \rceil$
apply(*subst picard-ivp.wp-g-orbit*[*of* $\lambda \ t \ s. \ c \ - \ 1/(t + 1) \ - \ (\lambda \ t \ x. \ x + t *_{\mathcal{R}} c)$])
using *constant-is-picard-ivp* **apply** *blast*
using *line-solves-constant* **by** *auto*

6.2 Verification with differential invariants

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

6.2.1 Differential Weakening

thm *kcomp-def kcomp-prop le-fun-def*

theorem *DW*:

shows $wp \ (\{[x'=f]T \ S \ @ \ t0 \ \& \ G\}) \lceil Q \rceil = wp \ (\{[x'=f]T \ S \ @ \ t0 \ \& \ G\}) \lceil \lambda \ s. \ G \ s \longrightarrow Q \ s \rceil$

unfolding *fbox-def* **apply**(*rule nd-fun-ext*) **apply** *transfer* **apply** *simp*

proof(*subst kcomp-prop*)**+**

fix $x::'a$ **and** $T \ f \ S \ t0 \ G \ Q$

let $?Y = g\text{-orbital} \ f \ T \ S \ t0 \ x \ G$

have $*\forall y \in ?Y. \ G \ y$ **by** *blast*

{assume $(\bigcup y \in ?Y. \ \text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\}$

then have $\forall y \in ?Y. \ (\text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\}$ **by** *blast*

hence $\forall y \in ?Y. \ Q \ y$ **by** (*metis* (*mono-tags*, *lifting*) *bot-pres-del*)

then have $\forall y \in ?Y. \ (\text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\}$ **by** *auto*

from this have $(\bigcup y \in ?Y. \ \text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\}$ **by** *blast*

moreover

{assume $(\bigcup y \in ?Y. \ \text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\}$

then have $\exists y \in ?Y. \ (\text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\}$ **by** *blast*

hence $\exists y \in ?Y. \ \neg Q \ y$ **by** (*metis* (*mono-tags*, *lifting*))

then have $\exists y \in ?Y. \ (\text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\}$

by (*metis* (*mono-tags*, *lifting*) ** bot-pres-del*)

from this have $(\bigcup y \in ?Y. \ \text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\}$ **by** *blast*

ultimately show $((\bigcup y \in ?Y. \ \text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\} \longrightarrow$

$(\bigcup y \in ?Y. \ \text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) = \{\}) \wedge$

$((\bigcup y \in ?Y. \ \text{if } \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\} \longrightarrow$

$(\bigcup y \in ?Y. \ \text{if } G \ y \wedge \neg Q \ y \text{ then } \eta \ y \text{ else } \{\}) \neq \{\})$

by *blast*

qed

theorem *dWeakening*:

assumes $\lceil G \rceil \leq \lceil Q \rceil$

shows $\lceil P \rceil \leq wp \ (\{[x'=f]T \ S \ @ \ t0 \ \& \ G\}) \lceil Q \rceil$

using *assms* **apply**(*subst wp-nd-fun*)
by(*auto simp: le-fun-def*)

6.2.2 Differential Cut

lemma *wp-g-orbit-etaD*:

assumes *wp* ($\{[x'=f] T S @ t0 \ \& \ G\}$) $\lceil C \rceil = \eta^\bullet$ **and** $\forall r \in \{t0--t\}. x \ r \in$
g-orbital *f* *T S t0 a G*
shows $\forall r \in \{t0--t\}. C \ (x \ r)$

proof

fix *r* **assume** $r \in \{t0--t\}$

then have $x \ r \in$ *g-orbital* *f* *T S t0 a G*

using *assms*(2) **by** *blast*

also have $\forall y. y \in$ (*g-orbital* *f* *T S t0 a G*) $\longrightarrow C \ y$

using *assms*(1) *wp-nd-fun-etaD* **by** *fastforce*

ultimately show $C \ (x \ r)$ **by** *blast*

qed

theorem *DC*:

assumes *t0* $\in T$ **and** *interval* *T*

and *wp* ($\{[x'=f] T S @ t0 \ \& \ G\}$) $\lceil C \rceil = \eta^\bullet$

shows *wp* ($\{[x'=f] T S @ t0 \ \& \ G\}$) $\lceil Q \rceil =$ *wp* ($\{[x'=f] T S @ t0 \ \& \ \lambda s. G \ s \wedge$
C s\}) $\lceil Q \rceil$

proof(*rule-tac f=λ x. wp x* $\lceil Q \rceil$ **in** *HOL.arg-cong*, *rule nd-fun-ext*, *rule subset-antisym*,
simp-all)

fix *a*

show *g-orbital* *f* *T S t0 a G* \subseteq *g-orbital* *f* *T S t0 a* ($\lambda s. G \ s \wedge C \ s$)

proof

fix *b* **assume** $b \in$ *g-orbital* *f* *T S t0 a G*

then obtain *t::real* **and** *x* **where** $t \in T$ **and** *x-solves*:(*x solves-ode* *f*) *T S*

and

x t0 = *a* **and** *guard-x*:($\forall r \in \{t0--t\}. G \ (x \ r)$) **and** $a \in S$ **and** $b = x \ t$

using *assms*(1) **unfolding** *f2r-def* **by** *blast*

from *guard-x* **have** $\forall r \in \{t0--t\}. \forall \tau \in \{t0--r\}. G \ (x \ \tau)$

using *assms*(1) **by** (*metis contra-subsetD ends-in-segment*(1) *subset-segment*(1))

also have $\forall r \in \{t0--t\}. r \in T$

using *assms*(1,2) $\langle t \in T \rangle$ *interval.closed-segment-subset-domain* **by** *blast*

ultimately have $\forall r \in \{t0--t\}. x \ r \in$ *g-orbital* *f* *T S t0 a G*

using *x-solves* $\langle x \ t0 = a \rangle \langle a \in S \rangle$ **unfolding** *f2r-def* **by** *blast*

from this have $\forall r \in \{t0--t\}. C \ (x \ r)$ **using** *wp-g-orbit-etaD* *assms*(3) **by**

blast

thus $b \in$ *g-orbital* *f* *T S t0 a* ($\lambda s. G \ s \wedge C \ s$) **unfolding** *f2r-def*

using *guard-x* $\langle a \in S \rangle \langle b = x \ t \rangle \langle t \in T \rangle \langle x \ t0 = a \rangle$ *x-solves* $\langle \forall r \in \{t0--t\}. r$

$\in T \rangle$ **by** *fastforce*

qed

next show $\bigwedge a. g\text{-orbital } f \ T \ S \ t0 \ a \ (\lambda s. G \ s \wedge C \ s) \subseteq g\text{-orbital } f \ T \ S \ t0 \ a \ G$ **by**
auto

qed

theorem *dCut*:

assumes $t0 \in T$ and interval T
 and $wp\text{-}C: [P] \leq wp (\{[x'=f] T S @ t0 \ \& \ G\}) [C]$
 and $wp\text{-}Q: [P] \leq wp (\{[x'=f] T S @ t0 \ \& \ (\lambda s. G s \wedge C s)\}) [Q]$
 shows $[P] \leq wp (\{[x'=f] T S @ t0 \ \& \ G\}) [Q]$
proof(subst wp-nd-fun, clarsimp)
 fix $t::real$ and $x::real \Rightarrow 'a$ assume $P (x t0)$ and $t \in T$ and $x t0 \in S$
 and $x\text{-solves}:(x \text{ solves-ode } f) T S$ and $guard\text{-}x:(\forall r \in \{t0--t\}. G (x r))$
 from $guard\text{-}x$ have $\forall r \in \{t0--t\}. \forall \tau \in \{t0--r\}. G (x \tau)$
 using $\langle t0 \in T \rangle$ by (metis contra-subsetD ends-in-segment(1) subset-segment(1))

 also have $\forall r \in \{t0--t\}. r \in T$
 using $\langle t0 \in T \rangle \langle interval \ T \rangle \langle t \in T \rangle interval.closed\text{-}segment\text{-}subset\text{-}domain$ by blast
 ultimately have $\forall r \in \{t0--t\}. x r \in g\text{-orbital } f \ T \ S \ t0 \ (x t0) \ G$
 using $x\text{-solves} \ \langle x t0 \in S \rangle$ by blast
 from this have $\forall r \in \{t0--t\}. C (x r)$ using $wp\text{-}C \ \langle P (x t0) \rangle$ by (subst (asm) wp-nd-fun, simp)
 hence $x t \in g\text{-orbital } f \ T \ S \ t0 \ (x t0) \ (\lambda s. G s \wedge C s)$
 using $guard\text{-}x \ \langle t \in T \rangle \ x\text{-solves} \ \langle x t0 \in S \rangle \ \langle \forall r \in \{t0--t\}. r \in T \rangle$ by fastforce
 from this $\langle P (x t0) \rangle$ and $wp\text{-}Q$ show $Q (x t)$
 by (subst (asm) wp-nd-fun, simp)
qed

corollary *dCut-interval*:

assumes $t0 \leq t$ and $[P] \leq wp (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) [C]$
 and $[P] \leq wp (\{[x'=f] \{t0..t\} S @ t0 \ \& \ (\lambda s. G s \wedge C s)\}) [Q]$
 shows $[P] \leq wp (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) [Q]$
apply(rule-tac $C=C$ in *dCut*)
 using *assms* by (simp-all add: interval-def)

6.2.3 Differential Invariant

lemma *DI-sufficiency*:

assumes *picard-ivp* $f \ T \ S \ L \ t0$
 shows $wp \ \{[x'=f] T S @ t0 \ \& \ G\} [Q] \leq wp [G] [\lambda s. s \in S \longrightarrow Q s]$
apply(subst wp-nd-fun, subst wp-nd-fun, clarsimp)
apply(erule-tac $x=s$ in *allE*, erule *impE*, rule-tac $x=t0$ in *exI*, simp-all)
 using *assms* *picard-ivp.fixed-point-solves* *picard-ivp.init-time* by metis

lemma

fixes $\vartheta::'a::banach \Rightarrow real$
 assumes $[G] \leq [I']$ and $t \geq 0$
 and $\forall x. (x \text{ solves-ode } f) \{0..t\} S \longrightarrow I (x 0) \longrightarrow$
 $(\forall t \geq 0. (\forall r \in \{0--t\}. I' (x r)) \longrightarrow (I (x t)))$
 shows $[I] \leq wp (\{[x'=f] \{0..t\} S @ 0 \ \& \ G\}) [I]$
 using *assms* **apply**(subst wp-nd-fun)
apply(subst *le-p2ndf-iff*) **apply** clarify
apply(erule-tac $x=x$ in *allE*)

apply(erule impE, simp)+
apply(erule-tac x=ta in allE)
by simp

definition pderivative :: 'a pred \Rightarrow 'a pred \Rightarrow (real \Rightarrow ('a::real-normed-vector) \Rightarrow 'a) \Rightarrow real set \Rightarrow
'a set \Rightarrow bool ((-)/ is'-pderivative'-of (-)/ with'-respect'-to (-) (-) (-) [70, 65] 61)
where
I' is-pderivative-of I with-respect-to f T S \equiv bdd-below T \wedge ($\forall x. (x \text{ solves-ode } f) T S \longrightarrow$
I (x (Inf T)) \longrightarrow ($\forall t \in T. (\forall r \in \{(Inf T) - - t\}. I' (x r)) \longrightarrow (I (x t))$))

lemma dInvariant:

assumes $\lceil G \rceil \leq \lceil I \rceil$ **and** I' is-pderivative-of I with-respect-to f T S
shows $\lceil I \rceil \leq wp \ (\{[x'=f] T S @ (Inf T) \ \& \ G\}) \ \lceil I \rceil$
using assms **unfolding** pderivative-def **apply**(subst wp-nd-fun)
apply(subst le-p2ndf-iff)
apply(clarify) **by** simp

lemma invariant-eq-0:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes nuHyp: $\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf T) - - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x r))) \text{ (at } r \text{ within } \{(Inf T) - - t\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu s = 0 \rceil$ **and** bdd-below T
shows $\lceil \lambda s. \vartheta s = 0 \rceil \leq wp \ (\{[x'=f] T S @ (Inf T) \ \& \ G\}) \ \lceil \lambda s. \vartheta s = 0 \rceil$
apply(rule dInvariant [of - $\lambda s. \nu s = 0$])
using assms **apply**(simp, simp add: pderivative-def)
proof(clarify)
fix x **and** t
assume x-ivp: $(x \text{ solves-ode } f) T S \vartheta (x (Inf T)) = 0$
and tHyp: $t \in T$ **and** eq0: $\forall r \in \{(Inf T) - - t\}. \nu (x r) = 0$
hence $(Inf T) \leq t$ **by** (simp add: $\langle \text{bdd-below } T \rangle$ cInf-lower)
have $\forall r \in \{(Inf T) - - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x r)))$
(at r **within** $\{(Inf T) - - t\}$ **) using** nuHyp x-ivp(1) **and** tHyp **by** auto
then have $\forall r \in \{(Inf T) - - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R 0))$
(at r **within** $\{(Inf T) - - t\}$ **) using** eq0 **by** auto
then have $\exists r \in \{(Inf T) - - t\}. \vartheta (x t) - \vartheta (x (Inf T)) = (\lambda \tau. \tau *_R 0) (t - (Inf T))$
by(rule-tac closed-segment-mvt, auto simp: $\langle (Inf T) \leq t \rangle$)
thus $\vartheta (x t) = 0$
using x-ivp(2) **by** (metis right-minus-eq scale-zero-right)
qed

corollary invariant-eq-0-interval:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{t0..\tau\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu s = 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta s = 0 \rceil \leq wp \ (\{[x'=f] \{t0..t\} S @ t0 \ \& \ G\}) \ \lceil \lambda s. \vartheta s = 0 \rceil$

apply(*subgoal-tac* $\lceil \lambda s. \vartheta s = 0 \rceil \leq wp \ (\{[x'=f]\{t0..t\} S @ (Inf \ \{t0..t\}) \ \& \ G\})$
 $\lceil \lambda s. \vartheta s = 0 \rceil$)
apply(*subgoal-tac* $Inf \ \{t0..t\} = t0, \text{ simp}$)
using $\langle t0 \leq t \rangle$ **apply** *simp*
apply(*rule invariant-eq-0* [of - $\{t0..t\}$ - - ν])
using *assms* **by**(*auto simp: closed-segment-eq-real-ivl*)

theorem *dInvariant-eq-0*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$ **and** $\nu::'a \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \ \{t0..t\} \ S \longrightarrow$
 $(\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta \ (x \ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu \ (x \ r)))) \ (at \ r$
within $\{t0..\tau\})$
and *impls*: $\lceil P \rceil \leq \lceil \lambda s. \vartheta s = 0 \rceil \ \lceil \lambda s. \vartheta s = 0 \rceil \leq \lceil Q \rceil \ \lceil G \rceil \leq \lceil \lambda s. \nu s = 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \leq wp \ (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ \lceil Q \rceil$
apply(*rule-tac* $C=\lambda s. \vartheta s = 0$ **in** *dCut-interval*, *simp add:* $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s = 0 \rceil \leq wp \ (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \ \lceil \lambda s. \vartheta s$
 $= 0 \rceil$)
using *impls* **apply**(*subst* (*asm*) *wp-nd-fun*, *subst wp-nd-fun*) **apply** *auto*[1]
apply(*rule-tac* $\nu=\nu$ **in** *invariant-eq-0-interval*)
using *assms*(1,4,5) **apply**(*simp*, *simp*, *simp*)
apply(*rule dWeakening*) **using** *impls* **by** *auto*

lemma *invariant-geq-0*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $nuHyp:\forall x. (x \text{ solves-ode } f) \ T \ S \longrightarrow (\forall t \in T. \forall r \in \{(Inf \ T)---t\}.$
 $((\lambda \tau. \vartheta \ (x \ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu \ (x \ r)))) \ (at \ r \text{ within } \{(Inf \ T)---t\})$
and $\lceil G \rceil \leq \lceil \lambda s. (\nu \ s) \geq 0 \rceil$ **and** *bdd-below* T
shows $\lceil \lambda s. \vartheta s \geq 0 \rceil \leq wp \ (\{[x'=f] \ T \ S @ (Inf \ T) \ \& \ G\}) \ \lceil \lambda s. \vartheta s \geq 0 \rceil$
apply(*rule dInvariant* [of - $\lambda s. \nu s \geq 0$])
using *assms* **apply**(*simp*, *simp add: pderivative-def*)
proof(*clarify*)
fix x **and** t
assume $x\text{-ivp}:\vartheta \ (x \ (Inf \ T)) \geq 0 \ (x \text{ solves-ode } f) \ T \ S$
and $tHyp:t \in T$ **and** $ge0:\forall r \in \{(Inf \ T)---t\}. \nu \ (x \ r) \geq 0$
hence $(Inf \ T) \leq t$ **by** (*simp add:* $\langle bdd\text{-below } T \rangle$ *cInf-lower*)
have $\forall r \in \{(Inf \ T)---t\}. ((\lambda \tau. \vartheta \ (x \ \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu \ (x \ r)))$
 $(at \ r \text{ within } \{(Inf \ T)---t\})$ **using** $nuHyp \ x\text{-ivp}(2)$ **and** $tHyp$ **by** *auto*
then have $\exists r \in \{(Inf \ T)---t\}. \vartheta \ (x \ t) - \vartheta \ (x \ (Inf \ T)) = (\lambda \tau. \tau *_R \nu \ (x \ r)) \ (t$
 $- (Inf \ T))$
by(*rule-tac closed-segment-mvt*, *auto simp:* $\langle (Inf \ T) \leq t \rangle$)
from this obtain r **where**
 $r \in \{(Inf \ T)---t\} \wedge \vartheta \ (x \ t) = (t - Inf \ T) *_R \nu \ (x \ r) + \vartheta \ (x \ (Inf \ T))$ **by** *force*

thus $0 \leq \vartheta \ (x \ t)$ **by** (*simp add:* $\langle Inf \ T \leq t \rangle$ *ge0 x-ivp(1)*)

qed

corollary *invariant-geq-0-interval*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$

assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{t0..\tau\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu s \geq 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta s \geq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \geq 0 \rceil$
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \geq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ (Inf \{t0..t\}) \ \& \ G\})$
 $\lceil \lambda s. \vartheta s \geq 0 \rceil$)
apply(*subgoal-tac* $Inf \{t0..t\} = t0, \text{ simp}$)
using $\langle t0 \leq t \rangle$ **apply**(*simp add: closed-segment-eq-real-ivl*)
apply(*rule invariant-geq-0* [*of* - $\{t0..t\}$ - - ν])
using *assms* **by**(*auto simp: closed-segment-eq-real-ivl*)

theorem *dInvariant-geq-0*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$ **and** $\nu::'a \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R \nu (x r)))) \text{ (at } r \text{ within } \{t0..\tau\}))$
and *impls*: $\lceil P \rceil \leq \lceil \lambda s. \vartheta s \geq 0 \rceil \lceil \lambda s. \vartheta s \geq 0 \rceil \leq \lceil Q \rceil \lceil G \rceil \leq \lceil \lambda s. \nu s \geq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil Q \rceil$
apply(*rule-tac* $C = \lambda s. \vartheta s \geq 0$ **in** *dCut-interval*, *simp add: t0 ≤ t*)
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \geq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \geq 0 \rceil$
 ≥ 0)
using *impls* **apply**(*subst (asm) wp-nd-fun, subst wp-nd-fun*) **apply** *auto*[1]
apply(*rule-tac* $\nu = \nu$ **in** *invariant-geq-0-interval*)
using *assms*(1,4,5) **apply**(*simp, simp, simp*)
apply(*rule dWeakening*) **using** *impls* **by** *auto*

lemma *invariant-leq-0*:

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $\text{nuHyp}:\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf T) - - t\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{(Inf T) - - t\}))$
and $\lceil G \rceil \leq \lceil \lambda s. (\nu s) \leq 0 \rceil$ **and** *bdd-below* T
shows $\lceil \lambda s. \vartheta s \leq 0 \rceil \leq wp (\{[x'=f] T S @ (Inf T) \ \& \ G\}) \lceil \lambda s. \vartheta s \leq 0 \rceil$
apply(*rule dInvariant* [*of* - $\lambda s. \nu s \leq 0$])
using *assms* **apply**(*simp, simp add: pderivative-def*)
proof(*clarify*)
fix x **and** t
assume $x\text{-ivp}:\vartheta (x (Inf T)) \leq 0 \text{ (} x \text{ solves-ode } f) T S$
and $t\text{Hyp}:\forall t \in T \text{ and } ge0:\forall r \in \{(Inf T) - - t\}. \nu (x r) \leq 0$
hence $(Inf T) \leq t$ **by** (*simp add: bdd-below T cInf-lower*)
have $\forall r \in \{(Inf T) - - t\}. ((\lambda\tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda\tau. \tau *_R (\nu (x r))))$
 $\text{ (at } r \text{ within } \{(Inf T) - - t\})$ **using** *nuHyp x-ivp(2)* **and** *tHyp* **by** *auto*
then have $\exists r \in \{(Inf T) - - t\}. \vartheta (x t) - \vartheta (x (Inf T)) = (\lambda\tau. \tau *_R (\nu (x r))) (t - (Inf T))$
by(*rule-tac closed-segment-mvt, auto simp: (Inf T) ≤ t*)
from this obtain r **where**
 $r \in \{(Inf T) - - t\} \wedge \vartheta (x t) = (t - Inf T) *_R \nu (x r) + \vartheta (x (Inf T))$ **by force**
thus $\vartheta (x t) \leq 0$ **using** $\langle (Inf T) \leq t \rangle$ *ge0 x-ivp(1)*
by (*metis add-decreasing2 ge-iff-diff-ge-0 split-scaleR-neg-le*)

qed

corollary *invariant-leq-0-interval*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{t0..\tau\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu s \leq 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta s \leq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \leq 0 \rceil$
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \leq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ (Inf \{t0..t\}) \ \& \ G\})$
 $\lceil \lambda s. \vartheta s \leq 0 \rceil$)
apply(*subgoal-tac* $Inf \{t0..t\} = t0, \text{ simp}$)
using $\langle t0 \leq t \rangle$ **apply**(*simp add: closed-segment-eq-real-ivl*)
apply(*rule invariant-leq-0[of - \{t0..t\} - - \nu]*)
using *assms* **by**(*auto simp: closed-segment-eq-real-ivl*)

theorem *dInvariant-leq-0*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$ **and** $\nu :: 'a \Rightarrow \text{real}$
assumes $\forall x. (x \text{ solves-ode } f) \{t0..t\} S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R \nu (x r))) \text{ (at } r \text{ within } \{t0..\tau\}))$
and *impls*: $\lceil P \rceil \leq \lceil \lambda s. \vartheta s \leq 0 \rceil \lceil \lambda s. \vartheta s \leq 0 \rceil \leq \lceil Q \rceil \lceil G \rceil \leq \lceil \lambda s. \nu s \leq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil Q \rceil$
apply(*rule-tac* $C = \lambda s. \vartheta s \leq 0$ **in** *dCut-interval, simp add: \langle t0 \leq t \rangle*)
apply(*subgoal-tac* $\lceil \lambda s. \vartheta s \leq 0 \rceil \leq wp (\{[x'=f]\{t0..t\} S @ t0 \ \& \ G\}) \lceil \lambda s. \vartheta s \leq 0 \rceil$
 $\leq 0 \rceil$)
using *impls* **apply**(*subst (asm) wp-nd-fun, subst wp-nd-fun*) **apply** *auto[1]*
apply(*rule-tac* $\nu = \nu$ **in** *invariant-leq-0-interval*)
using *assms(1,4,5)* **apply**(*simp, simp, simp*)
apply(*rule dWeakening*) **using** *impls* **by** *auto*

lemma *invariant-above-0*:

fixes $\vartheta :: 'a :: \text{banach} \Rightarrow \text{real}$
assumes *nuHyp*: $\forall x. (x \text{ solves-ode } f) T S \longrightarrow (\forall t \in T. \forall r \in \{(Inf T) - - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r)))) \text{ (at } r \text{ within } \{(Inf T) - - t\}))$
and $\lceil G \rceil \leq \lceil \lambda s. (\nu s) \geq 0 \rceil$ **and** *bdd-below* T
shows $\lceil \lambda s. \vartheta s > 0 \rceil \leq wp (\{[x'=f] T S @ (Inf T) \ \& \ G\}) \lceil \lambda s. \vartheta s > 0 \rceil$
apply(*rule dInvariant [of - \lambda s. \nu s \geq 0]*)
using *assms* **apply**(*simp, simp add: pderivative-def*)
proof(*clarify*)
fix x **and** t
assume $x\text{-ivp}: (x \text{ solves-ode } f) T S \vartheta (x (Inf T)) > 0$
and *tHyp*: $t \in T$ **and** *ge0*: $\forall r \in \{(Inf T) - - t\}. \nu (x r) \geq 0$
hence $(Inf T) \leq t$ **by** (*simp add: \langle bdd-below T \rangle cInf-lower*)
have $\forall r \in \{(Inf T) - - t\}. ((\lambda \tau. \vartheta (x \tau)) \text{ has-derivative } (\lambda \tau. \tau *_R (\nu (x r))))$
 $\text{ (at } r \text{ within } \{(Inf T) - - t\})$ **using** *nuHyp x-ivp(1)* **and** *tHyp* **by** *auto*
then have $\exists r \in \{(Inf T) - - t\}. \vartheta (x t) - \vartheta (x (Inf T)) = (\lambda \tau. \tau *_R (\nu (x r))) (t - (Inf T))$
by(*rule-tac closed-segment-mvt, auto simp: \langle (Inf T) \leq t \rangle*)

from this obtain r where
 $r \in \{(Inf\ T) \dashv\dashv t\} \wedge \vartheta\ (x\ t) = (t - Inf\ T) *_R \nu\ (x\ r) + \vartheta\ (x\ (Inf\ T))$ **by force**
thus $0 < \vartheta\ (x\ t)$
by (metis (Inf\ T) $\leq t$ ge0 x-ivp(2) Groups.add-ac(2) add-mono-thms-linordered-field(3)
 $ge\text{-}iff\text{-}diff\text{-}ge\text{-}0\ monoid\text{-}add\text{-}class.add\text{-}0\text{-}right\ scaleR\text{-}nonneg\text{-}nonneg)$
qed

corollary invariant-above-0-interval:

fixes $\vartheta::'a::banach \Rightarrow real$
assumes $\forall x. (x\ solves\ ode\ f)\{t0..t\}\ S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has\ derivative\ (\lambda\tau. \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu\ s \geq 0 \rceil$ and $t0 \leq t$
shows $\lceil \lambda s. \vartheta\ s > 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil \lambda s. \vartheta\ s > 0 \rceil$
apply(subgoal-tac $\lceil \lambda s. \vartheta\ s > 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ (Inf\ \{t0..t\}) \ \&\ G\})$
 $\lceil \lambda s. \vartheta\ s > 0 \rceil$ **)**
apply(subgoal-tac $Inf\ \{t0..t\} = t0$, simp)
using $\langle t0 \leq t \rangle$ apply(simp add: closed-segment-eq-real-ivl)
apply(rule invariant-above-0[of - $\{t0..t\}$ - - ν])
using assms by(auto simp: closed-segment-eq-real-ivl)

theorem dInvariant-above-0:

fixes $\vartheta::'a::banach \Rightarrow real$ and $\nu::'a \Rightarrow real$
assumes $\forall x. (x\ solves\ ode\ f)\{t0..t\}\ S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..\tau\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has\ derivative\ (\lambda\tau. \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
and impls: $\lceil P \rceil \leq \lceil \lambda s. \vartheta\ s > 0 \rceil\ \lceil \lambda s. \vartheta\ s > 0 \rceil \leq \lceil Q \rceil\ \lceil G \rceil \leq \lceil \lambda s. \nu\ s \geq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil Q \rceil$
apply(rule-tac $C = \lambda s. \vartheta\ s > 0$ in dCut-interval, simp add: $\langle t0 \leq t \rangle$)
apply(subgoal-tac $\lceil \lambda s. \vartheta\ s > 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil \lambda s. \vartheta\ s > 0 \rceil$
 $> 0 \rceil$ **)**
using impls apply(subst (asm) wp-nd-fun, subst wp-nd-fun) apply auto[1]
apply(rule-tac $\nu = \nu$ in invariant-above-0-interval)
using assms(1,4,5) apply(simp, simp, simp)
apply(rule dWeakening) using impls by auto

lemma invariant-below-0:

fixes $\vartheta::'a::banach \Rightarrow real$
assumes nuHyp: $\forall x. (x\ solves\ ode\ f)\ T\ S \longrightarrow (\forall t \in T. \forall r \in \{(Inf\ T) \dashv\dashv t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has\ derivative\ (\lambda\tau. \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{(Inf\ T) \dashv\dashv t\}))$
and $\lceil G \rceil \leq \lceil \lambda s. (\nu\ s) \leq 0 \rceil$ and bdd-below T
shows $\lceil \lambda s. \vartheta\ s < 0 \rceil \leq wp\ (\{[x'=f]\ T\ S @ (Inf\ T) \ \&\ G\})\ \lceil \lambda s. \vartheta\ s < 0 \rceil$
apply(rule dInvariant [of - $\lambda s. \nu\ s \leq 0$])
using assms apply(simp, simp add: pderivative-def)
proof(clarify)
fix x and t
assume x-ivp: $(x\ solves\ ode\ f)\ T\ S\ \vartheta\ (x\ (Inf\ T)) < 0$

and $tHyp: t \in T$ **and** $ge0: \forall r \in \{Inf\ T - - t\}. \nu\ (x\ r) \leq 0$
hence $(Inf\ T) \leq t$ **by** (*simp add: <bdd-below T> cInf-lower*)
have $\forall r \in \{(Inf\ T) - - t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_{\mathbb{R}} (\nu\ (x\ r))))$
(at r within {(Inf T) - - t}) **using** *nuHyp x-ivp(1)* **and** *tHyp* **by** *auto*
then have $\exists r \in \{(Inf\ T) - - t\}. \vartheta\ (x\ t) - \vartheta\ (x\ (Inf\ T)) = (\lambda\tau. \tau *_{\mathbb{R}} (\nu\ (x\ r)))\ (t$
 $- (Inf\ T))$
by (*rule-tac closed-segment-mvt, auto simp: <(Inf T) ≤ t>*)
thus $\vartheta\ (x\ t) < 0$ **using** $(Inf\ T) \leq t$ *ge0 x-ivp(2)*
by (*metis add-mono-thms-linordered-field(3) diff-gt-0-iff-gt ge-iff-diff-ge-0 linorder-not-le*
monoid-add-class.add-0-left monoid-add-class.add-0-right split-scaleR-neg-le)

qed

corollary *invariant-below-0-interval:*

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x\ solves-ode\ f)\{t0..t\}\ S \longrightarrow (\forall \tau \in \{t0..t\}. \forall r \in \{t0..t\}.$
 $((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_{\mathbb{R}} (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..t\}))$
and $\lceil G \rceil \leq \lceil \lambda s. \nu\ s \leq 0 \rceil$ **and** $t0 \leq t$
shows $\lceil \lambda s. \vartheta\ s < 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil \lambda s. \vartheta\ s < 0 \rceil$
apply (*subgoal-tac* $\lceil \lambda s. \vartheta\ s < 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ (Inf\ \{t0..t\}) \ \&\ G\})$
 $\lceil \lambda s. \vartheta\ s < 0 \rceil$)
apply (*subgoal-tac* $Inf\ \{t0..t\} = t0$, *simp*)
using $\langle t0 \leq t \rangle$ **apply** (*simp add: closed-segment-eq-real-ivl*)
apply (*rule invariant-below-0[of - {t0..t} - - ν]*)
using *assms* **by** (*auto simp: closed-segment-eq-real-ivl*)

theorem *dInvariant-below-0:*

fixes $\vartheta::'a::\text{banach} \Rightarrow \text{real}$
assumes $\forall x. (x\ solves-ode\ f)\{t0..t\}\ S \longrightarrow$
 $(\forall \tau \in \{t0..t\}. \forall r \in \{t0..t\}. ((\lambda\tau. \vartheta\ (x\ \tau))\ has-derivative\ (\lambda\tau. \tau *_{\mathbb{R}} \nu\ (x\ r))))\ (at\ r$
within {t0..t})
and *impls:* $\lceil P \rceil \leq \lceil \lambda s. \vartheta\ s < 0 \rceil\ \lceil \lambda s. \vartheta\ s < 0 \rceil \leq \lceil Q \rceil\ \lceil G \rceil \leq \lceil \lambda s. \nu\ s \leq 0 \rceil$
and $t0 \leq t$
shows $\lceil P \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil Q \rceil$
using $\langle t0 \leq t \rangle$ **apply** (*rule-tac* $C = \lambda s. \vartheta\ s < 0$ **in** *dCut-interval*, *simp add: <t0 ≤ t>*)
apply (*subgoal-tac* $\lceil \lambda s. \vartheta\ s < 0 \rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S @ t0 \ \&\ G\})\ \lceil \lambda s. \vartheta\ s$
 $< 0 \rceil$)
using *impls* **apply** (*subst (asm) wp-nd-fun, subst wp-nd-fun*) **apply** *auto[1]*
apply (*rule-tac* $\nu = \nu$ **in** *invariant-below-0-interval*)
using *assms(1,4,5)* **apply** (*simp, simp, simp*)
apply (*rule dWeakening*) **using** *impls* **by** *auto*

lemma *invariant-meet:*

assumes $\lceil I1 \rceil \leq wp\ (\{[x'=f]\ T\ S @ t0 \ \&\ G\})\ \lceil I1 \rceil$
and $\lceil I2 \rceil \leq wp\ (\{[x'=f]\ T\ S @ t0 \ \&\ G\})\ \lceil I2 \rceil$
shows $\lceil \lambda s. I1\ s \wedge I2\ s \rceil \leq wp\ (\{[x'=f]\ T\ S @ t0 \ \&\ G\})\ \lceil \lambda s. I1\ s \wedge I2\ s \rceil$
using *assms* **by** (*subst (asm) wp-nd-fun, subst (asm) wp-nd-fun, subst wp-nd-fun,*

simp, blast)

theorem *dInvariant-meet*:

assumes $\lceil I1 \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I1 \rceil$ and $\lceil I2 \rceil \leq wp$
 $(\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I2 \rceil$
and *impls*: $\lceil P \rceil \leq \lceil \lambda s. \ I1 \ s \wedge \ I2 \ s \rceil \ \lceil \lambda s. \ I1 \ s \wedge \ I2 \ s \rceil \leq \lceil Q \rceil$ and $t0 \leq t$
shows $\lceil P \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil Q \rceil$
apply(*rule-tac* $C=\lambda s. \ I1 \ s \wedge \ I2 \ s$ **in** *dCut-interval*, *simp add*: $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $\lceil \lambda s. \ I1 \ s \wedge \ I2 \ s \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil \lambda s.$
 $I1 \ s \wedge \ I2 \ s \rceil$)
using *impls* **apply**(*transfer*, *simp add*: *le-fun-def*) **apply** *auto*[1]
apply(*rule invariant-meet*)
using *assms*(1,2,5) **apply**(*simp*, *simp*)
apply(*rule dWeakening*)
using *impls* **by** *simp*

lemma *invariant-join*:

assumes $\lceil I1 \rceil \leq wp \ (\{[x'=f] \ T \ S \ @ \ t0 \ \& \ G\}) \ \lceil I1 \rceil$
and $\lceil I2 \rceil \leq wp \ (\{[x'=f] \ T \ S \ @ \ t0 \ \& \ G\}) \ \lceil I2 \rceil$
shows $\lceil \lambda s. \ I1 \ s \vee \ I2 \ s \rceil \leq wp \ (\{[x'=f] \ T \ S \ @ \ t0 \ \& \ G\}) \ \lceil \lambda s. \ I1 \ s \vee \ I2 \ s \rceil$
using *assms* **by**(*subst* (*asm*) *wp-nd-fun*, *subst* (*asm*) *wp-nd-fun*, *subst wp-nd-fun*,
simp)

theorem *dInvariant-join*:

assumes $\lceil I1 \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I1 \rceil$ and $\lceil I2 \rceil \leq wp$
 $(\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil I2 \rceil$
and *impls*: $\lceil P \rceil \leq \lceil \lambda s. \ I1 \ s \vee \ I2 \ s \rceil \ \lceil \lambda s. \ I1 \ s \vee \ I2 \ s \rceil \leq \lceil Q \rceil$ and $t0 \leq t$
shows $\lceil P \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil Q \rceil$
apply(*rule-tac* $C=\lambda s. \ I1 \ s \vee \ I2 \ s$ **in** *dCut-interval*, *simp add*: $\langle t0 \leq t \rangle$)
apply(*subgoal-tac* $\lceil \lambda s. \ I1 \ s \vee \ I2 \ s \rceil \leq wp \ (\{[x'=f]\{t0..t\} \ S \ @ \ t0 \ \& \ G\}) \ \lceil \lambda s.$
 $I1 \ s \vee \ I2 \ s \rceil$)
using *impls* **apply**(*transfer*, *simp add*: *le-fun-def*) **apply** *auto*[1]
apply(*rule invariant-join*)
using *assms*(1,2,5) **apply**(*simp*, *simp*)
apply(*rule dWeakening*)
using *impls* **by** *auto*

end

theory *cat2funcset-examples*

imports *cat2funcset*

begin

6.3 Examples

Here we do our first verification example: the single-evolution ball. We do it in two ways. The first one provides (1) a finite type and (2) its corresponding problem-specific vector-field and flow. The second approach uses an existing finite type and defines a more general vector-field which is later instantiated

to the problem at hand.

6.3.1 Specific vector field

We define a finite type of three elements. All the lemmas below proven about this type must exist in order to do the verification example.

```
typedef three = {m::nat. m < 3}
apply(rule-tac x=0 in exI)
by simp
```

```
lemma CARD-of-three: CARD(three) = 3
using type-definition.card type-definition-three by fastforce
```

```
instance three::finite
apply(standard, subst bij-betw-finite[of Rep-three UNIV {m::nat. m < 3}])
apply(rule bij-betwI')
apply (simp add: Rep-three-inject)
using Rep-three apply blast
apply (metis Abs-three-inverse UNIV-I)
by simp
```

```
lemma three-univD:(UNIV::three set) = {Abs-three 0, Abs-three 1, Abs-three 2}
proof –
  have (UNIV::three set) = Abs-three ‘ {m::nat. m < 3}
    apply auto by (metis Rep-three Rep-three-inverse image-iff)
  also have {m::nat. m < 3} = {0, 1, 2} by auto
  ultimately show ?thesis by auto
qed
```

```
lemma three-exhaust:∀ x::three. x = Abs-three 0 ∨ x = Abs-three 1 ∨ x =
Abs-three 2
using three-univD by auto
```

Next we use our recently created type to generate a 3-dimensional vector space. We then define the vector field and the flow for the single-evolution ball on this vector space. Then we follow the standard procedure to prove that they are in fact a Lipschitz vector-field and a its flow.

```
abbreviation free-fall-kinematics (s::real^three) ≡ (χ i. if i=(Abs-three 0) then s
$ (Abs-three 1) else
if i=(Abs-three 1) then s $ (Abs-three 2) else 0)
```

```
abbreviation free-fall-flow t s ≡
(χ i. if i=(Abs-three 0) then s $ (Abs-three 2) · t ^ 2/2 + s $ (Abs-three 1) · t +
s $ (Abs-three 0)
else if i=(Abs-three 1) then s $ (Abs-three 2) · t + s $ (Abs-three 1) else s $
(Abs-three 2))
```

```
lemma bounded-linear-free-fall-kinematics:bounded-linear free-fall-kinematics
```

```

apply unfold-locales
  apply(simp-all add: plus-vec-def scaleR-vec-def ext norm-vec-def L2-set-def)
apply(rule-tac x=1 in exI, clarsimp)
apply(subst three-univD, subst three-univD)
by(auto simp: Abs-three-inject)

lemma free-fall-kinematics-continuous-on: continuous-on X free-fall-kinematics
  using bounded-linear-free-fall-kinematics linear-continuous-on by blast

lemma free-fall-kinematics-is-picard-ivp: 0 ≤ t ⇒ t < 1 ⇒
picard-ivp (λ t s. free-fall-kinematics s) {0..t} UNIV 1 0
  unfolding picard-ivp-def picard-ivp-axioms-def ubc-definitions
  apply(simp-all add: nonempty-set-def lipschitz-on-def, safe)
  apply(rule continuous-on-compose2[of UNIV - {0..t} × UNIV snd])
  apply(simp-all add: free-fall-kinematics-continuous-on continuous-on-snd)
  apply(simp add: dist-vec-def L2-set-def dist-real-def)
  apply(subst three-univD, subst three-univD)
  by(simp add: Abs-three-inject)

lemma free-fall-flow-solves-free-fall-kinematics:
  ((λ τ. free-fall-flow τ s) solves-ode (λ t s. free-fall-kinematics s) {0..t} UNIV
  apply (rule solves-vec-lambda)
  apply(simp add: solves-ode-def)
  unfolding has-vderiv-on-def has-vector-derivative-def apply(auto simp: Abs-three-inject)
  using poly-derivatives(3, 4) unfolding has-vderiv-on-def has-vector-derivative-def
  by auto)

We end the first example by computing the wlp of the kinematics for the
single-evolution ball and then using it to verify "its safety".

corollary free-fall-flow-DS:
  assumes 0 ≤ t and t < 1
  shows wp {[x' = λ t s. free-fall-kinematics s]{0..t} UNIV @ 0 & G} [Q] =
[λ x. ∀ τ ∈ {0..t}. (∀ r ∈ {0..τ}. G (free-fall-flow r x)) → Q (free-fall-flow
τ x)]
  apply(subst picard-ivp.wp-g-orbit[of λ t s. free-fall-kinematics s - - 1 - (λ t x.
free-fall-flow t x)])
  using free-fall-kinematics-is-picard-ivp and assms apply blast apply(clarify,
rule conjI)
  using free-fall-flow-solves-free-fall-kinematics apply blast
  apply(simp add: vec-eq-iff) using three-exhaust by auto

lemma single-evolution-ball:
  assumes 0 ≤ t and t < 1
  shows
  [λ s. (0::real) ≤ s $ (Abs-three 0) ∧ s $ (Abs-three 0) = H ∧ s $ (Abs-three 1) =
0 ∧ 0 > s $ (Abs-three 2)]
  ≤ wp ({[x' = λ t s. free-fall-kinematics s]{0..t} UNIV @ 0 & (λ s. s $ (Abs-three
0) ≥ 0)})
  [λ s. 0 ≤ s $ (Abs-three 0) ∧ s $ (Abs-three 0) ≤ H]

```

```

apply(subst free-fall-flow-DS)
by(simp-all add: asms mult-nonneg-nonpos2)

```

6.3.2 General vector field

It turns out that there is already a 3-element type:

```

term x::3
lemma CARD(three) = CARD(3)
unfolding CARD-of-three by simp

```

In fact, for each natural number n there is already a corresponding n -element type in Isabelle. However, there are still some lemmas that one needs to prove in order to use it in verification in n -dimensional vector spaces.

lemma *exhaust-5*: — The analog for 3 has already been proven in Analysis.

```

fixes x::5
shows x=1  $\vee$  x=2  $\vee$  x=3  $\vee$  x=4  $\vee$  x=5
proof (induct x)
case (of-int z)
then have  $0 \leq z$  and  $z < 5$  by simp-all
then have  $z = 0 \vee z = 1 \vee z = 2 \vee z = 3 \vee z = 4$  by arith
then show ?case by auto
qed

```

```

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

```

```

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

```

Next, we prove that every linear system of differential equations (i.e. it can be rewritten as $x' = A \cdot x$) satisfies the conditions of the Picard-Lindelöf theorem:

```

lemma matrix-lipschitz-constant:
fixes A::real^('n::finite)^'n
shows dist (A *v x) (A *v y)  $\leq$  (real CARD('n))2 · maxAbs A · dist x y
unfolding dist-norm vector-norm-distr-minus proof(subst norm-matrix-sgn)
have normS A  $\leq$  maxAbs A · (real CARD('n) · real CARD('n))
by (metis (no-types) Groups.mult-ac(2) norms-le-dims-maxAbs)
then have normS A · norm (x - y)  $\leq$  (real CARD('n))2 · maxAbs A · norm
(x - y)
by (simp add: cross3-simps(11) mult-left-mono semiring-normalization-rules(29))
also have norm (A *v sgn (x - y)) · norm (x - y)  $\leq$  normS A · norm (x - y)
by (simp add: norm-sgn-le-norms cross3-simps(11) mult-left-mono)
ultimately show norm (A *v sgn (x - y)) · norm (x - y)  $\leq$  (real CARD('n))2
· maxAbs A · norm (x - y)
using order-trans-rules(23) by blast

```

qed

lemma *picard-ivp-linear-system*:

fixes $A::\text{real}^{('n::\text{finite})} \rightarrow 'n$

assumes $0 < ((\text{real CARD}('n))^2 \cdot (\text{maxAbs } A))$ (**is** $0 < ?L$)

assumes $0 \leq t$ **and** $t < 1/?L$

shows *picard-ivp* $(\lambda t s. A * v s) \{0..t\} \text{ UNIV } ?L 0$

apply *unfold-locales*

subgoal by (*simp, metis continuous-on-compose2 continuous-on-cong continuous-on-id*

continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest)

subgoal using *matrix-lipschitz-constant maxAbs-ge-0 zero-compare-simps(4,12)*

unfolding *lipschitz-on-def* **by** *blast*

apply (*simp-all add: assms*)

subgoal for $r s$ **apply** (*subgoal-tac* $|r - s| < 1/((\text{real CARD}('n))^2 \cdot \text{maxAbs } A)$)

apply (*subst (asm) pos-less-divide-eq* [*of ?L* $|r - s| 1$])

using *assms* **by** *auto*

done

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

lemma *axis* $(1::3) (1::\text{real}) = (\chi j. \text{if } j = 0 \text{ then } 0 \text{ else if } j = 1 \text{ then } 1 \text{ else } 0)$

unfolding *axis-def* **by** (*rule Cart-lambda-cong, simp*)

abbreviation $K \equiv (\chi i. \text{if } i = (0::3) \text{ then } \text{axis } (1::3) (1::\text{real}) \text{ else if } i = 1 \text{ then } \text{axis } 2 \ 1 \text{ else } 0)$

abbreviation *flow-for-K* $t s \equiv (\chi i. \text{if } i = (0::3) \text{ then } s \$ 2 \cdot t \wedge 2/2 + s \$ 1 \cdot t + s \$ 0$

else if $i=1$ *then* $s \$ 2 \cdot t + s \$ 1 \text{ else } s \$ 2)$

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

lemma *entries-K*: *entries* $K = \{0, 1\}$

apply (*simp-all add: axis-def, safe*)

by (*rule-tac x=1 in exI, simp*)**+**

lemma *K-is-picard-ivp*: $0 \leq t \implies t < 1/9 \implies$

picard-ivp $(\lambda t s. K * v s) \{0..t\} \text{ UNIV } ((\text{real CARD}(3))^2 \cdot \text{maxAbs } K) 0$

apply (*rule picard-ivp-linear-system*)

unfolding *entries-K* **by** *auto*

lemma *flow-for-K-solves-K*: $((\lambda \tau. \text{flow-for-K } \tau s) \text{ solves-ode } (\lambda t s. K * v s)) \{0..t\} \text{ UNIV}$

apply (*rule solves-vec-lambda*)

apply (*simp add: solves-ode-def*)

using *poly-derivatives(1, 3, 4)*

by(*auto simp: matrix-vector-mult-def*)

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

corollary *flow-for-K-DS*:

assumes $0 \leq t$ **and** $t < 1/9$
shows $wp \{[x' = \lambda t \ s. K * v \ s] \{0..t\} \ UNIV @ 0 \ \& \ G\} \lceil Q \rceil =$
 $\lceil \lambda \ x. \ \forall \ \tau \in \{0..t\}. \ (\forall r \in \{0..-\tau\}. \ G \ (flow\text{-}for\text{-}K \ r \ x)) \longrightarrow Q \ (flow\text{-}for\text{-}K \ \tau \ x) \rceil$
apply(*subst picard-ivp.wp-g-orbit[of $\lambda t \ s. K * v \ s - ((real \ CARD(\mathcal{I}))^2 \cdot maxAbs \ K) - (\lambda t \ x. flow\text{-}for\text{-}K \ t \ x)$]*)
using *K-is-picard-ivp* **and** *assms* **apply** *blast* **apply**(*clarify, rule conjI*)
using *flow-for-K-solves-K* **apply** *blast*
apply(*simp add: vec-eq-iff*) **using** *exhaust-3* **apply** *force*
by *simp*

lemma *single-evolution-ball-K*:

assumes $0 \leq t$ **and** $t < 1/9$
shows $\lceil \lambda s. (0::real) \leq s \ \$ \ (0::\mathcal{I}) \wedge s \ \$ \ 0 = H \wedge s \ \$ \ 1 = 0 \wedge 0 > s \ \$ \ 2 \rceil$
 $\leq wp \ (\{[x' = \lambda t \ s. K * v \ s] \{0..t\} \ UNIV @ 0 \ \& \ (\lambda s. s \ \$ \ 0 \geq 0)\})$
 $\lceil \lambda s. 0 \leq s \ \$ \ 0 \wedge s \ \$ \ 0 \leq H \rceil$
apply(*subst flow-for-K-DS*)
using *assms* **by**(*simp-all add: mult-nonneg-nonpos2*)

6.3.3 Bouncing Ball with solution

Armed now with two vector fields for free-fall kinematics and their respective flows, proving the safety of a “bouncing ball” is merely an exercise of real arithmetic:

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

lemma [*bb-real-arith*]: $0 \leq x \implies 0 > g \implies 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \implies (x::real) \leq H$

proof–

assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$

then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*

hence $*:v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$

using *left-diff-distrib* *mult.commute* **by** (*metis zero-le-square*)

from this have $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*

also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$

using *divide-nonneg-neg* **by** *fastforce*

ultimately have $H - x \geq 0$ **by** *linarith*

thus *?thesis* **by** *auto*

qed

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$

and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$

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

Groups.mult-ac(2, 3) *monoid-mult-class.power2-eq-square* *nat-distrib*(2))
 hence $2 \cdot g \cdot H + (- ((g \cdot \tau) + v))^2 = 0$
 by (*metis* *Groups.add-ac*(2) *power2-minus*)
 thus *?thesis*
 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 $\dots = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$ (**is** $\dots = ?middle$)
 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 $\dots = ?middle$
 by (*simp add: semiring-normalization-rules*(29))
 finally have $?rhs = ?middle$.}
 ultimately show *?thesis* **by** *auto*
qed

lemma [*wp* (*IF* ($\lambda s. s \ \$ \ 0 = 0$) *THEN* ($(\lambda s. \eta \ (s[1 := - \ s \ \$ \ 1]))^\bullet$) *ELSE* η^\bullet *FI*)
 $\lceil \lambda s. 0 \leq s \ \$ \ 0 \wedge s \ \$ \ 2 < 0 \wedge 2 \cdot s \ \$ \ 2 \cdot s \ \$ \ 0 = 2 \cdot s \ \$ \ 2 \cdot H + s \ \$ \ 1 \cdot s \ \$ \ 1 \rceil$
 $= Q$
 apply(*subst wp-trafo*) **thm** *wp-trafo*
oops

lemma *bouncing-ball*:
 assumes $0 \leq t$ and $t < 1/9$
 shows $\lceil \lambda s. (0::real) \leq s \ \$ \ (0::3) \wedge s \ \$ \ 0 = H \wedge s \ \$ \ 1 = 0 \wedge 0 > s \ \$ \ 2 \rceil \leq wp$
 $((\{[x' = \lambda t \ s. K * v \ s] \{0..t\} \ UNIV \ @ \ 0 \ \& \ (\lambda s. s \ \$ \ 0 \geq 0)\} \ .$

```

(IF (λ s. s $ 0 = 0) THEN ([1 ::= (λ s. - s $ 1)]) ELSE η• FI))*)
[λ s. 0 ≤ s $ 0 ∧ s $ 0 ≤ H]
apply(subst star-nd-fun.abs-eq, rule rel-ad-mka-starI [of - [λ s. 0 ≤ s $ (0::3) ∧
0 > s $ 2 ∧
2 · s $ 2 · s $ 0 = 2 · s $ 2 · H + (s $ 1 · s $ 1)])])
apply(simp, simp only: fbox-seq)
apply(subst p2ndf-ndf2p-wp-sym[of (IF (λ s. s $ 0 = 0) THEN ([1 ::= (λ s.
- s $ 1)]) ELSE η• FI)])
apply(subst flow-for-K-DS) using assms apply(simp, simp)
oops

```

6.3.4 Bouncing Ball with invariants

```

lemma gravity-is-invariant:(x solves-ode (λ t. ( *v) K)) {0..t} UNIV ⇒ τ ∈
{0..t} ⇒ r ∈ {0..τ} ⇒
((λ τ. x τ $ 2) has-derivative (λ τ. τ *R 0)) (at r within {0..τ})
apply(drule-tac i=2 in solves-vec-nth)
apply(unfold solves-ode-def has-vderiv-on-def has-vector-derivative-def, clarify)
apply(erule-tac x=r in ballE, simp add: matrix-vector-mult-def)
by (simp-all add: has-derivative-within-subset)

```

```

lemma bouncing-ball-invariant:(x solves-ode (λ t. ( *v) K)) {0..t} UNIV ⇒ τ ∈
{0..t} ⇒
r ∈ {0..τ} ⇒ ((λ τ. 2 · x τ $ 2 · x τ $ 0 - 2 · x τ $ 2 · H - x τ $ 1 · x τ $
1) has-derivative
(λ τ. τ *R 0)) (at r within {0..τ})
apply(frule-tac i=2 in solves-vec-nth, frule-tac i=1 in solves-vec-nth, drule-tac
i=0 in solves-vec-nth)
apply(unfold solves-ode-def has-vderiv-on-def has-vector-derivative-def, clarify)
apply(erule-tac x=r in ballE, simp-all add: matrix-vector-mult-def)+
apply(rule-tac f'1=λ t. 2 · x r $ 2 · (t · x r $ 1)
and g'1=λ t. 2 · (t · (x r $ 1 · x r $ 2)) in derivative-eq-intros(11))
apply(rule-tac f'1=λ t. 2 · x r $ 2 · (t · x r $ 1) and g'1=λ t. 0 in
derivative-eq-intros(11))
apply(rule-tac f'1=λ t. 0 and g'1=(λ x a. x a · x r $ 1) in derivative-eq-intros(12))
apply(rule-tac g'1=λ t. 0 in derivative-eq-intros(6), simp-all add: has-derivative-within-subset)
apply(rule-tac g'1=λ t. 0 in derivative-eq-intros(7))
apply(rule-tac g'1=λ t. 0 in derivative-eq-intros(6), simp-all add: has-derivative-within-subset)
by(rule-tac f'1=(λ x a. x a · x r $ 2) and g'1=(λ x a. x a · x r $ 2) in derivative-eq-intros(12),

simp-all add: has-derivative-within-subset)

```

lemma bouncing-ball-invariants:

```

assumes 0 ≤ t and t < 1/9
shows [λ s. (0::real) ≤ s $ (0::3) ∧ s $ 0 = H ∧ s $ 1 = 0 ∧ 0 > s $ 2] ≤ wp
((([x'=λ t s. K *v s]{0..t} UNIV @ 0 & (λ s. s $ 0 ≥ 0)) ·
(IF (λ s. s $ 0 = 0) THEN ([1 ::= (λ s. - s $ 1)]) ELSE η• FI))*)
[λ s. 0 ≤ s $ 0 ∧ s $ 0 ≤ H]
apply(subst star-nd-fun.abs-eq,

```

```

rule-tac I=[ $\lambda s. 0 \leq s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1)$ ]
in rel-ad-mka-starI)
  apply(simp, simp only: fbox-seq)
  apply(subst p2ndf-ndf2p-wp-sym[of (IF ( $\lambda s. s \$ 0 = 0$ ) THEN ( $[1 ::= (\lambda s. s \$ 1)]$ ) ELSE  $\eta^\bullet FI$ )]])
  using assms(1) apply(rule dCut-interval[of - - - -  $\lambda s. s \$ 2 < 0$ ])
  apply(rule-tac  $\vartheta = \lambda s. s \$ 2$  and  $\nu = \lambda s. 0$  in dInvariant-below-0)
  using gravity-is-invariant apply force
  apply(simp, simp, simp, simp add:  $\langle 0 \leq t \rangle$ )
  apply(rule-tac  $C = \lambda s. 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$  in dCut-interval, simp add:  $\langle 0 \leq t \rangle$ )
  apply(rule-tac  $\vartheta = \lambda s. 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1$  and  $\nu = \lambda s. 0$  in dInvariant-eq-0)
  using bouncing-ball-invariant apply force
  apply(simp, simp, simp, simp add:  $\langle 0 \leq t \rangle$ )
  apply(rule dWeakening, subst p2ndf-ndf2p-wp)
oops

```

6.3.5 Circular motion with invariants

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

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

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

lemma sum-axis-UNIV-2[simp]: $(\sum j \in (UNIV::2 \text{ set}). \text{axis } i \text{ } r \$ j \cdot f j) = r \cdot (f::2 \Rightarrow \text{real}) \text{ } i$
 unfolding axis-def UNIV-2 by simp

abbreviation Circ $\equiv (\chi \text{ } i. \text{if } i = (0::2) \text{ then axis } (1::2) \text{ } (-1::\text{real}) \text{ else axis } 0 \text{ } 1)$

abbreviation flow-for-Circ $t \text{ } s \equiv (\chi \text{ } i. \text{if } i = (0::2) \text{ then } s\$0 \cdot \cos t - s\$1 \cdot \sin t \text{ else } s\$0 \cdot \sin t + s\$1 \cdot \cos t)$

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

lemma Circ-is-picard-ivp: $0 \leq t \Longrightarrow t < 1/4 \Longrightarrow$
 picard-ivp $(\lambda t \text{ } s. \text{Circ } * v \text{ } s) \{0..t\} \text{ UNIV } ((\text{real } \text{CARD}(2))^2 \cdot \text{maxAbs } \text{Circ}) \text{ } 0$
 apply(rule picard-ivp-linear-system)
 unfolding entries-Circ by auto

lemma flow-for-Circ-solves-Circ: $((\lambda \tau. \text{flow-for-Circ } \tau \text{ } s) \text{ solves-ode } (\lambda t \text{ } s. \text{Circ } * v \text{ } s)) \{0..t\} \text{ UNIV}$


```

apply (rule solves-vec-lambda, clarsimp)
subgoal for  $i$  apply (cases  $i=0$ )
  apply (simp-all add: matrix-vector-mult-def)
unfolding solves-ode-def has-vderiv-on-def has-vector-derivative-def apply auto
subgoal for  $x$ 
  apply (rule-tac  $f'1=\lambda t. - s\$0 \cdot (t \cdot \sin x)$  and  $g'1=\lambda t. s\$1 \cdot (t \cdot \cos x)$ ) in
  derivative-eq-intros(11))
  apply (rule derivative-eq-intros(6)[of  $\cos (\lambda x a. - (x a \cdot \sin x))$ ])
  apply (rule-tac  $Db1=1$  in derivative-eq-intros(58))
  apply (rule ssubst[of  $(\cdot) 1$  id], force, simp, force, force)
  apply (rule derivative-eq-intros(6)[of  $\sin (\lambda x a. (x a \cdot \cos x))$ ])
  apply (rule-tac  $Db1=1$  in derivative-eq-intros(55))
  apply (rule ssubst[of  $(\cdot) 1$  id], force, simp, force, force)
by (simp add: Groups.mult-ac(3) Rings.ring-distrib(4))
subgoal for  $x$ 
  apply (rule-tac  $f'1=\lambda t. s\$0 \cdot (t \cdot \cos x)$  and  $g'1=\lambda t. - s\$1 \cdot (t \cdot \sin x)$ ) in
  derivative-eq-intros(8))
  apply (rule derivative-eq-intros(6)[of  $\sin (\lambda x a. x a \cdot \cos x)$ ])
  apply (rule-tac  $Db1=1$  in derivative-eq-intros(55))
  apply (rule ssubst[of  $(\cdot) 1$  id], force, simp, force, force)
  apply (rule derivative-eq-intros(6)[of  $\cos (\lambda x a. - (x a \cdot \sin x))$ ])
  apply (rule-tac  $Db1=1$  in derivative-eq-intros(58))
  apply (rule ssubst[of  $(\cdot) 1$  id], force, simp, force, force)
by (simp add: Groups.mult-ac(3) Rings.ring-distrib(4))
done
done

```

corollary flow-for-Circ-DS:

```

assumes  $0 \leq t$  and  $t < 1/4$ 
shows  $wp \{[x'=\lambda t s. \text{Circ } *v s]\{0..t\} \text{ UNIV } @ 0 \ \& \ G\} [Q] =$ 
 $[\lambda x. \forall \tau \in \{0..t\}. (\forall r \in \{0..-\tau\}. G (\text{flow-for-Circ } r x)) \longrightarrow Q (\text{flow-for-Circ } \tau x)]$ 
apply (subst picard-ivp.wp-g-orbit[of  $\lambda t s. \text{Circ } *v s - ((\text{real CARD}(2))^2 \cdot \max\text{-Abs Circ}) - (\lambda t x. \text{flow-for-Circ } t x)$ ])
using Circ-is-picard-ivp and assms apply blast apply (clarify, rule conjI)
using flow-for-Circ-solves-Circ apply blast
apply (simp add: vec-eq-iff) using exhaust-2 two-eq-zero apply force
by simp

```

lemma circular-motion:

```

assumes  $0 \leq t$  and  $t < 1/4$  and  $(R::\text{real}) > 0$ 
shows  $[\lambda s. R^2 = (s \$ (0..2))^2 + (s \$ 1)^2] \leq wp$ 
 $\{[x'=\lambda t s. \text{Circ } *v s]\{0..t\} \text{ UNIV } @ 0 \ \& \ (\lambda s. s \$ 0 \geq 0)\}$ 
 $[\lambda s. R^2 = (s \$ (0..2))^2 + (s \$ 1)^2]$ 
apply (subst flow-for-Circ-DS)
using assms by simp-all

```

lemma circular-motion-invariants:

```

assumes  $0 \leq t$  and  $t < 1/4$  and  $(R::\text{real}) > 0$ 

```

```

shows[ $\lambda s. R^2 = (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2$ ]  $\leq wp$ 
{[ $x' = \lambda t \ s. \text{Circ} \ *v \ s\} \{0..t\} \text{UNIV} \ @ \ 0 \ \& \ (\lambda \ s. \ s \ \$ \ 0 \geq 0)$ }]
[ $\lambda s. R^2 = (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2$ ] thm dCut-interval
using assms(1) apply(rule-tac  $C = \lambda s. R^2 = (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2$  in dCut-interval,
simp)
apply(subgoal-tac ( $\lambda s. (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2 - R^2 = 0$ ) = ( $\lambda s. R^2 = (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2$ ))
apply(rule ssubst[of ( $\lambda s. R^2 = (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2$ )  $\lambda s. (s \ \$ \ (0::2))^2 + (s \ \$ \ 1)^2 - R^2 = 0$ ], simp)
apply(rule-tac  $\vartheta = \lambda s. (s \ \$ \ 0)^2 + (s \ \$ \ 1)^2 - R^2$  and  $\nu = \lambda s. 0$  in dInvariant-eq-0)
subgoal apply clarify
apply(frule-tac  $i=0$  in solves-vec-nth, drule-tac  $i=1$  in solves-vec-nth)
apply(unfold solves-ode-def has-vderiv-on-def has-vector-derivative-def, clar-simp)
apply(erule-tac  $x=r$  in ballE, simp-all add: matrix-vector-mult-def)+
sorry
apply(simp, simp, simp, simp add: (0 ≤ t)) apply auto[1]
by(rule dWeakening, simp)

end

```