CPSVerification

CPSVerification

February 11, 2019

Contents

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* ($\lceil$-$\rceil$)
  and *Archimedean-Field.floor* ($\lfloor$-$\rfloor$)
  and *Set.image* (‘)
  and *Range-Semiring.antirange-semiring-class.ars-r* (r)

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

This and more notation is explained by the following lemmata.

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

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

Observe also, the following consequences and facts:

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

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

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

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

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

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

proposition *rel-ad-rule2*: $(x,x) \in \Delta^c{}_1\ \lceil P \rceil \Longrightarrow \neg\ P\ x$

by(*metis ComplD VC-KAD.p2r-neg-hom rel-ad-rule1 empty-iff mem-Collect-eq p2s-neg-hom*

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

proposition *rel-ad-rule3*: $R \subseteq Id \Longrightarrow (x,x) \notin R \Longrightarrow (x,x) \in \Delta^{c}{}_{1}\ R$
by(*metis IdI Un-iff d-p2r rel-antidomain-kleene-algebra.addual.ars3*
rel-antidomain-kleene-algebra.addual.ars-r-def rpr)

proposition *rel-ad-rule4*: $(x,x) \in R \Longrightarrow (x,x) \notin \Delta^{c}{}_{1}\ R$
by(*metis empty-iff rel-antidomain-kleene-algebra.addual.ars1 relcomp.relcompI*)

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

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

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

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

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

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

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

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

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

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

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

proposition *vdiff-invarDiffs*:(∂ *x*) ∈ *varDiffs*
by (*simp add*: *varDiffsI*)

1.2.1 (primed) dSolve preliminaries

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

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

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

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

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

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

primrec *state-list-upd* :: ((*real* ⇒ *real store* ⇒ *real*) × *string* × (*real store* ⇒ *real*)) *list* ⇒
real ⇒ *real store* ⇒ *real store* **where**
state-list-upd [] *t s* = *s*|
state-list-upd (*uxf* # *tail*) *t s* = (*state-list-upd tail t s*)

($(\pi_1\ (\pi_2\ uxf)) := (\pi_1\ uxf)\ t\ s$,
$\partial\ (\pi_1\ (\pi_2\ uxf)) := (if\ t = 0\ then\ (\pi_2\ (\pi_2\ uxf))\ s$
$else\ vderiv\text{-}of\ (\lambda\ r.\ (\pi_1\ uxf)\ r\ s)\ \{0{<}..{<}\ (2 *_R t)\}\ t))$

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

lemma *inductive-state-list-cross-upd-its-vars*:
assumes *distHyp*:*distinct* (*map* π_1 ((*y*, *g*) # *xftail*))
and *varHyp*:$\forall$ *xf*$\in$*set*((*y*, *g*) # *xftail*). π_1 *xf* $\notin$ *varDiffs*
and *indHyp*:(*u*, *x*, *f*) $\in$ *set* (*utail* $\otimes$ *xftail*) $\Longrightarrow$ (*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*,
  ∂ *x* := *if t* = *0 then f s else vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $*_R$ *t*)} *t*)) *x* **by** *simp*
  also have ... = *u t s* **by** (*simp add*: *vdiff-def*)
  ultimately show *?thesis* **by** *simp*
next
  assume *yTailHyp*:(*u*, *x*, *f*) $\in$ *set* (*utail* $\otimes$ *xftail*)
  from *this* **and** *indHyp* **have** *3*:(*s*[*xftail*$\leftarrow$*utail*] *t*) *x* = *u t s* **by** *fastforce*
  from *yTailHyp* **and** *distHyp* **have** *2*:*y* $\neq$ *x* **using** *set-zip-left-rightD* **by** *force*
  from *yTailHyp* **and** *varHyp* **have** *1*:*x* $\neq$ ∂ *y*
  using *set-zip-left-rightD vdiff-invarDiffs* **by** *fastforce*
  from *1* **and** *2* **have** (*s*[(*y*, *g*) # *xftail*$\leftarrow$*v* # *utail*] *t*) *x* = (*s*[*xftail*$\leftarrow$*utail*] *t*) *x* **by** *simp*
  thus *?thesis* **using** *3* **by** *simp*
qed

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

lemma *override-on-upd*:$x \in X \Longrightarrow (override\text{-}on\ f\ g\ X)(x := z) = (override\text{-}on\ f\ (g(x := z))\ X)$

by (*rule ext*, *simp add*: *override-on-def*)

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

theorem *state-list-cross-upd-its-dvars*:
assumes *lengthHyp*:$length\ xfList = length\ uInput$
and *varsHyp*:$\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *solHyp1*:$\forall uxf \in set\ (uInput \otimes xfList).\ (\pi_1\ uxf)\ 0\ s = s\ (\pi_1\ (\pi_2\ uxf))$
shows $\exists\ g.\ (s[xfList \leftarrow uInput]\ 0) = (override\text{-}on\ s\ g\ varDiffs)$
using *assms* **proof**(*induct xfList uInput rule*: *list-induct2'*)
case *1*
  have $(s[[] \leftarrow []]\ 0) = override\text{-}on\ s\ s\ varDiffs$
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *metis*
next
  case (*2 xf xfTail*)
  have $(s[(xf\ \#\ xfTail) \leftarrow []]\ 0) = override\text{-}on\ s\ s\ varDiffs$
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *metis*
next
  case (*3 u utail*)
  have $(s[[] \leftarrow utail]\ 0) = override\text{-}on\ s\ s\ varDiffs$
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *force*
next
  case (*4 xf xfTail u uTail*)
  then have $\exists g.\ (s[xfTail \leftarrow uTail]\ 0) = override\text{-}on\ s\ g\ varDiffs$ **by** *simp*
  thus *?case* **using** *inductive-state-list-cross-upd-its-dvars 4.prems* **by** *blast*
qed

lemma *vderiv-unique-within-open-interval*:
assumes (*f has-vderiv-on f'*) $\{0{<}..{<}\ t\}$ **and** $t>0$

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

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

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

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

lemma *inductive-to-sol-zero-its-dvars*:
assumes *eqFuncs*:$\forall$ *s*. $\forall$ *g*. $\forall$ *xf*$\in$*set* ((*x*, *f*) # *xfs*). π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *eqLengths*:*length* ((*x*, *f*) # *xfs*) = *length* (*u* # *us*)
and *distinct*:*distinct* (*map* π_1 ((*x*, *f*) # *xfs*))
and *vars*:$\forall$ *xf*$\in$*set* ((*x*, *f*) # *xfs*). π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* ((*u* # *us*) $\otimes$ ((*x*, *f*) # *xfs*)). π_1 *uxf 0* (*sol s*) = *sol s* (π_1 (π_2 *uxf*))
and *disjHyp*:(*y*, *g*) = (*x*, *f*) $\vee$ (*y*, *g*) $\in$ *set xfs*
and *indHyp*:(*y*, *g*) $\in$ *set xfs* $\Longrightarrow$ (*sol s*[*xfs*$\leftarrow$*us*] *0*) (∂ *y*) = *g* (*sol s*[*xfs*$\leftarrow$*us*] *0*)
shows (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*)
proof−
from *assms* **obtain** *h1* **where** *h1Def*:(*sol s*[((*x*, *f*) # *xfs*)$\leftarrow$(*u* # *us*)] *0*) = (*override-on* (*sol s*) *h1 varDiffs*) **using** *state-list-cross-upd-its-dvars* **by** *blast*
from *disjHyp* **show** (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*)
proof
  assume *eqHeads*:(*y*, *g*) = (*x*, *f*)
  then have *g* (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*) = *f* (*sol s*) **using** *h1Def eqFuncs* **by** *simp*
  also have ... = (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*) (∂ *y*) **using** *eqHeads* **by** *auto*
  ultimately show *?thesis* **by** *linarith*
next
  assume *tailHyp*:(*y*, *g*) $\in$ *set xfs*
  then have *y* $\neq$ *x* **using** *distinct set-zip-left-rightD* **by** *force*
  hence ∂ *x* $\neq$ ∂ *y* **by**(*simp add*: *vdiff-def*)
  have *x* $\neq$ ∂ *y* **using** *vars vdiff-invarDiffs* **by** *auto*
  obtain *h2* **where** *h2Def*:(*sol s*[*xfs*$\leftarrow$*us*] *0*) = *override-on* (*sol s*) *h2 varDiffs*
  using *state-list-cross-upd-its-dvars eqLengths distinct vars* **and** *solHyp1* **by** *force*
  have (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[*xfs*$\leftarrow$*us*] *0*)
  using *tailHyp indHyp* ‹*x* $\neq$ ∂ *y*› **and** ‹∂ *x* $\neq$ ∂ *y*› **by** *simp*
  also have ... = *g* (*override-on* (*sol s*) *h2 varDiffs*) **using** *h2Def* **by** *simp*
  also have ... = *g* (*sol s*) **using** *eqFuncs* **and** *tailHyp* **by** *force*
  also have ... = *g* (*sol s*[(*x*, *f*) # *xfs*$\leftarrow$*u* # *us*] *0*)
  using *eqFuncs h1Def tailHyp* **and** *eq-snd-iff* **by** *fastforce*
  ultimately show *?thesis* **by** *simp*
  qed
qed

lemma *to-sol-zero-its-dvars*:
assumes *funcsHyp*:$\forall$ *s*. $\forall$ *g*. $\forall$ *xf* $\in$ *set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf* $\in$ *set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *ygHyp*:(*y*, *g*) $\in$ *set xfList*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *0*)(∂ *y*) = *g* (*sol s*[*xfList*$\leftarrow$*uInput*] *0*)

using *assms* **apply**(*induct xfList uInput rule*: *list-induct2'*, *simp*, *simp*, *simp*, *clarify*)
by(*rule inductive-to-sol-zero-its-dvars*, *simp-all*)

lemma *inductive-to-sol-greater-than-zero-its-dvars*:
assumes *lengthHyp*:$length\ ((y,\ g)\ \#\ xfs) = length\ (v\ \#\ vs)$
and *distHyp*:$distinct\ (map\ \pi_1\ ((y,\ g)\ \#\ xfs))$
and *varHyp*: $\forall xf \in set\ ((y,\ g)\ \#\ xfs).\ \pi_1\ xf \notin varDiffs$
and *indHyp*:$(u,x,f) \in set\ (vs \otimes xfs) \Longrightarrow (s[xfs \leftarrow vs]t)(\partial\ x) = vderiv\text{-}of\ (\lambda r.\ u\ r\ s)\ \{0<..<2*_R t\}\ t$
and *disjHyp*:$(v,\ y,\ g) = (u,\ x,\ f) \vee (u,\ x,\ f) \in set\ (vs \otimes xfs)$ **and** *tHyp*:$t > 0$
shows $(s[(y,\ g)\ \#\ xfs \leftarrow v\ \#\ vs]\ t)\ (\partial\ x) = vderiv\text{-}of\ (\lambda r.\ u\ r\ s)\ \{0<..<2\ *_R\ t\}\ t$
proof−
let $?lhs = ((s[xfs \leftarrow vs]\ t)(y := v\ t\ s,\ \partial\ y := vderiv\text{-}of\ (\lambda\ r.\ v\ r\ s)\ \{0<..<\ (2\ \cdot\ t)\}\ t))\ (\partial\ x)$
let $?rhs = vderiv\text{-}of\ (\lambda r.\ u\ r\ s)\ \{0<..<\ (2\ \cdot\ t)\}\ t$
have $(s[(y,\ g)\ \#\ xfs \leftarrow v\ \#\ vs]\ t)\ (\partial\ x) = ?lhs$ **using** *tHyp* **by** *simp*
also have $vderiv\text{-}of\ (\lambda r.\ u\ r\ s)\ \{0<..<2\ *_R\ t\}\ t = ?rhs$ **by** *simp*
ultimately have *obs*:$?thesis = (?lhs = ?rhs)$ **by** *simp*
from *disjHyp* **have** $?lhs = ?rhs$
proof
  assume *uxfEq*:$(v,\ y,\ g) = (u,\ x,\ f)$
  then have $?lhs = vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0<..<\ (2\ \cdot\ t)\}\ t$ **by** *simp*
  also have $vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0<..<\ (2\ \cdot\ t)\}\ t = ?rhs$ **using** *uxfEq* **by** *simp*
  ultimately show $?lhs = ?rhs$ **by** *simp*
next
  assume *sygTail*:$(u,\ x,\ f) \in set\ (vs \otimes xfs)$
  from *this* **have** $y \neq x$ **using** *distHyp set-zip-left-rightD* **by** *force*
  hence $\partial\ x \neq \partial\ y$ **by**(*simp add*: *vdiff-def*)
  have $y \neq \partial\ x$ **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
  then have $?lhs = (s[xfs \leftarrow vs]\ t)\ (\partial\ x)$ **using** ‹$y \neq \partial\ x$› **and** ‹$\partial\ x \neq \partial\ y$› **by** *simp*
  also have $(s[xfs \leftarrow vs]\ t)\ (\partial\ x) = ?rhs$ **using** *indHyp sygTail* **by** *simp*
  ultimately show $?lhs = ?rhs$ **by** *simp*
qed
from *this* **and** *obs* **show** *?thesis* **by** *simp*
qed

lemma *to-sol-greater-than-zero-its-dvars*:
assumes *distinctHyp*:$distinct\ (map\ \pi_1\ xfList)$
and *lengthHyp*:$length\ xfList = length\ uInput$
and *varsHyp*:$\forall xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *uxfHyp*:$(u,\ x,\ f) \in set\ (uInput \otimes xfList)$ **and** *tHyp*:$t > 0$
shows $(s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0<..<\ (2\ *_R\ t)\}\ t$
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2'*, *simp*, *simp*, *simp*, *clarify*)
by(*rule-tac f=f* **in** *inductive-to-sol-greater-than-zero-its-dvars*, *auto*)

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*$[\![$ - $]\!]_t$)) **where**
$[\![t_C\ r]\!]_t = (\lambda\ s.\ r)|$
$[\![t_V\ x]\!]_t = (\lambda\ s.\ s\ x)|$
$[\![\ominus\ \vartheta]\!]_t = (\lambda\ s.\ -\ ([\![\vartheta]\!]_t)\ s)|$
$[\![\vartheta \oplus \eta]\!]_t = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s + ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \odot \eta]\!]_t = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s \cdot ([\![\eta]\!]_t)\ s)$

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

primrec *pval* ::*props* $\Rightarrow$ (*real store* $\Rightarrow$ *bool*) ((*1*$[\![$-$]\!]_P$)) **where**
$[\![\vartheta \doteq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s = ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \prec \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s < ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \preceq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s \leq ([\![\eta]\!]_t)\ s)|$
$[\![\varphi \sqcap \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \wedge ([\![\psi]\!]_P)\ s)|$
$[\![\varphi \sqcup \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \vee ([\![\psi]\!]_P)\ s)$

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

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

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

$trmVars\ (\vartheta \oplus \eta) = trmVars\ \vartheta \cup trmVars\ \eta|$
$trmVars\ (\vartheta \odot \eta) = trmVars\ \vartheta \cup trmVars\ \eta$

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

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

lemma *substList-help1*:$set\ (map\ \pi_1\ ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)) \subseteq varDiffs$
apply(*induct xfList uInput rule*: *list-induct2′*, *simp-all add*: *varDiffs-def*)
by *auto*

lemma *substList-help2*:
assumes $trmVars\ \eta \subseteq (UNIV - varDiffs)$
shows $((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\eta\rangle = \eta$
using *assms substList-help1 substList-on-compl-of-varDiffs* **by** *blast*

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

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

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

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 this theorems as instantiations.

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

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

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

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

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

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

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

lemma $t > 0 \Longrightarrow$ *vderiv-of* $(\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x)\ \{0{<}..{<}2 \cdot t\}\ t = a \cdot t + v$
unfolding *vderiv-of-def* **apply**(*subst some1-equality*[*of* - $(\lambda t.\ a \cdot t + v)$])
apply(*rule-tac* $a = \lambda t.\ a \cdot t + v$ **in** *ex1I*)
apply(*simp-all add*: *galilean-position*)
apply(*rule ext*, *rename-tac f* τ)
apply(*rule-tac* $f = \lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x$ **and** $t = 2 \cdot t$ **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*]:
$t > 0 \Longrightarrow$ *vderiv-of* $(\lambda r.\ a \cdot r + v)\ \{0{<}..{<}2 \cdot t\}\ t = a$

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

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

lemma [*galilean-transform-eq*]:*t* > *0* ⟹ *vderiv-of* (λ*t*. *v* · *t* − *a* · *t*ˆ*2* / *2* + *x*) {*0*<..<*2* · *t*} *t* = *v* − *a* · *t*
apply(*subgoal-tac vderiv-of* (λ*t*. − *a* · t^2 / *2* + *v* · *t* + *x*) {*0*<..<*2* · *t*} *t* = − *a* · *t* + *v*, *simp*)
by(*rule galilean-transform-eq*)

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

lemma [*galilean-transform-eq*]:*t* > *0* ⟹ *vderiv-of* (λ*r*. *v* − *a* · *r*) {*0*<..<*2* · *t*} *t* = − *a*
apply(*subgoal-tac vderiv-of* (λ*t*. − *a* · *t* + *v*) {*0*<..<*2* · *t*} *t* = − *a*, *simp*)
by(*rule galilean-transform-eq*)

lemma [*simp*]:(λ*x*. *case x of* (*t*, *x*) ⇒ *f t*) = (λ *x*. (*f* ∘ π_1) *x*)
by *auto*

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

begin

1.3 Phase Space Relational Semantics

definition *solvesStoreIVP* :: (*real* ⇒ *real store*) ⇒ (*string* × (*real store* ⇒ *real*)) *list* ⇒

real store ⇒ *bool*
((- *solvesTheStoreIVP* - *withInitState* -) [*70*, *70*, *70*] *68*) **where**
solvesStoreIVP φ_S *xfList s* ≡
— F sends vdiffs-in-list to derivs.
(∀ $t \geq 0$. (∀ *xf* ∈ *set xfList*. φ_S *t* (∂ (π_1 *xf*)) = π_2 *xf* (φ_S *t*)) ∧
— F preserves the rest of the variables and F sends derivs of constants to 0.
(∀ *y*. (*y* ∉ (π_1⦇*set xfList*⦈) ∪ *varDiffs* ⟶ φ_S *t y* = *s y*) ∧
(*y* ∉ (π_1⦇*set xfList*⦈) ⟶ φ_S *t* (∂ *y*) = *0*)) ∧
— F solves the induced IVP.
(∀ *xf* ∈ *set xfList*. ((λ *t*. φ_S *t* (π_1 *xf*)) *solves-ode* (λ *t*.λ *r*.(π_2 *xf*) (φ_S *t*))) {*0*..*t*} *UNIV* ∧
φ_S *0* (π_1 *xf*) = *s*(π_1 *xf*)))

lemma *solves-store-ivpI*:
assumes ∀ *t* ≥ *0*.∀ *xf* ∈ *set xfList*. (φ_S *t* (∂ (π_1 *xf*))) = (π_2 *xf*) (φ_S *t*)
and ∀ *t* ≥ *0*.∀ *y*. *y* ∉ (π_1⦇*set xfList*⦈)∪*varDiffs* ⟶ φ_S *t y* = *s y*
and ∀ *t* ≥ *0*.∀ *y*. *y* ∉ (π_1⦇*set xfList*⦈) ⟶ φ_S *t* (∂ *y*) = *0*
and ∀ *t* ≥ *0*. ∀ *xf* ∈ *set xfList*. ((λ *t*. φ_S *t* (π_1 *xf*)) *solves-ode* (λ *t*.λ *r*.(π_2 *xf*) (φ_S *t*))) {*0*..*t*} *UNIV*
and ∀ *xf* ∈ *set xfList*. φ_S *0* (π_1 *xf*) = *s*(π_1 *xf*)
shows φ_S *solvesTheStoreIVP xfList withInitState s*
apply(*simp add*: *solvesStoreIVP-def*, *safe*)
using *assms* **apply** *simp-all*
by(*force*,*force*,*force*)

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

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
shows ∀ *t* ≥ *0*.∀ *y*. *y* ∉ (π_1⦇*set xfList*⦈)∪*varDiffs* ⟶ φ_S *t y* = *s y*
and ∀ *t* ≥ *0*.∀ *y*. *y* ∉ (π_1⦇*set xfList*⦈) ⟶ φ_S *t* (∂ *y*) = *0*
and ∀ *t* ≥ *0*.∀ *xf* ∈ *set xfList*. (φ_S *t* (∂ (π_1 *xf*))) = (π_2 *xf*) (φ_S *t*)
and ∀ *t* ≥ *0*. ∀ *xf* ∈ *set xfList*. ((λ *t*. φ_S *t* (π_1 *xf*)) *solves-ode* (λ *t*.λ *r*.(π_2 *xf*) (φ_S *t*))) {*0*..*t*} *UNIV*
and ∀ *xf* ∈ *set xfList*. φ_S *0* (π_1 *xf*) = *s*(π_1 *xf*)
using *assms solvesStoreIVP-def* **by** *auto*

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
shows ∀ *y*. *y* ∉ *varDiffs* ⟶ φ_S *0 y* = *s y*
proof(*clarify*, *rename-tac x*)
fix *x* **assume** *x* ∉ *varDiffs*
from *assms* **and** *solves-store-ivpE*(*5*) **have** *x* ∈ (π_1⦇*set xfList*⦈) ⟹ φ_S *0 x* = *s x* **by** *fastforce*
also have *x* ∉ (π_1⦇*set xfList*⦈) ∪ *varDiffs* ⟹ φ_S *0 x* = *s x*
using *assms* **and** *solves-store-ivpE*(*1*) **by** *simp*
ultimately show φ_S *0 x* = *s x* **using** ‹*x* ∉ *varDiffs*› **by** *auto*
qed

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

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

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

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

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
  and $t \geq 0$
  and $xf \in set\ xfList$
shows $((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))$ *solves-ode* $(\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}\ UNIV$
using *assms solves-store-ivpE(4)* **by** *simp*

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

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

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

1.4 Derivation of Differential Dynamic Logic Rules

1.4.1 "Differential Weakening"

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

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

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

1.4.2 "Differential Cut"

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

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

theorem *dCut*:
assumes *pBoxDiffCut*:(*PRE P* (*ODEsystem xfList with G*) *POST C*)
assumes *pBoxCutQ*:(*PRE P* (*ODEsystem xfList with* ($\lambda\ s.\ G\ s \wedge C\ s$)) *POST Q*)
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*clarify*, *subgoal-tac a = b*) **defer**
proof(*metis d-p2r rdom-p2r-contents*, *simp*, *subst boxProgrPred-chrctrztn*, *clarify*)
fix *b y* **assume** $(b, b) \in \lceil P \rceil$ **and** $(b, y) \in$ *ODEsystem xfList with G*
then obtain φ_S *t* **where** $*$:*solvesStoreIVP* φ_S *xfList b* $\wedge$ ($\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge 0 \leq t \wedge \varphi_S\ t = y$
  using *guarDiffEqtn-def* **by** *auto*

hence $\forall\ r \in \{0..t\}.\ (b,\ \varphi_S\ r) \in (ODEsystem\ xfList\ with\ G)$
using *all-interval-guarDiffEqtn* **by** *blast*
from *this* **and** *pBoxDiffCut* **have** $\forall\ r \in \{0..t\}.\ C\ (\varphi_S\ r)$
using *boxProgrPred-chrctrztn* ‹$(b,\ b) \in \lceil P \rceil$› **by** (*metis* (*no-types*, *lifting*) *d-p2r subsetCE*)
then have $\forall\ r \in \{0..t\}.\ (b,\ \varphi_S\ r) \in (ODEsystem\ xfList\ with\ (\lambda\ s.\ G\ s \wedge C\ s))$
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-chrctrztn* ‹$(b,\ b) \in \lceil P \rceil$› **by** (*metis* (*no-types*, *lifting*) *d-p2r subsetCE*)
thus $Q\ y$ **using** $*$ **by** *auto*
qed

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

1.4.3 "Solve Differential Equation"

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

using *guarDiffEqtn-def* **by** *auto*
from *this* **and** *uniqHyp* **have** (*sol s*[*xfList*←*uInput*] *t*) = φ_S *t* **by** *blast*
then have *cHyp*:*c* = (*sol s*[*xfList*←*uInput*] *t*) **using** *FHyp* **by** *simp*
from *this* **have** *G* (*sol s*[*xfList*←*uInput*] *t*) **using** *FHyp* **by** *force*
then show *Q c* **using** *diffAssgn FHyp cHyp* **by** *auto*
qed

theorem *dS*:
assumes *solHyp*:∀ *s*. *solvesStoreIVP* (λ*t*. *sol s*[*xfList*←*uInput*] *t*) *xfList s*
and *uniqHyp*:∀ *s X*. *solvesStoreIVP X xfList s* ⟶ (∀ *t* ≥ *0*. (*sol s*[*xfList*←*uInput*] *t*) = *X t*)
shows *wp* (*ODEsystem xfList with G*) ⌈*Q*⌉ =
⌈λ *s*. ∀ *t*≥*0*. (∀ *r*∈{*0*..*t*}. *G* (*sol s*[*xfList*←*uInput*] *r*)) ⟶ *Q* (*sol s*[*xfList*←*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*=λ*t*. *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 *solHyp*:∀ *s*. *solvesStoreIVP* (λ*t*. *sol s*[*xfList*←*uInput*] *t*) *xfList s*
and *uniqHyp*:∀ *s*. ∀ *X*. *solvesStoreIVP X xfList s* ⟶ (∀ *t* ≥ *0*.(*sol s*[*xfList*←*uInput*] *t*) = *X t*)
and *diffAssgn*: ∀ *s*. *P s* ⟶ (∀ *t*≥*0*. *G* (*sol s*[*xfList*←*uInput*] *t*) ⟶ *Q* (*sol s*[*xfList*←*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*clarsimp*, *subgoal-tac a*=*b*)
apply(*clarify*, *subst boxProgrPred-chrctrztn*)
apply(*simp-all add*: *p2r-def*)
apply(*rule-tac uInput*=*uInput* **in** *prelim-dSolve*)
apply(*simp add*: *solHyp*, *simp add*: *uniqHyp*)
by (*metis* (*no-types*, *lifting*) *diffAssgn*)

— We proceed to refine the previous rule by finding the necessary restrictions on varFunList and uInput so that the solution to the store-IVP is guaranteed.

lemma *conds4vdiffs-prelim*:
assumes *funcsHyp*:∀ *s g*. ∀ *xf*∈*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:∀ *uxf* ∈ *set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))

and *solHyp2*:∀ *t*≥*0*. ((λτ. (*sol s*[*xfList*←*uInput*] τ) *x*)
has-vderiv-on (λτ. *f* (*sol s*[*xfList*←*uInput*] τ))) {*0*..*t*}
and *xfHyp*:(*x*, *f*) ∈ *set xfList* **and** *tHyp*:*t* ≥ *0*
shows (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *t*)
proof−
from *xfHyp* **obtain** *u* **where** *xfuHyp*: (*u*,*x*,*f*) ∈ *set* (*uInput* ⊗ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
show (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) =*f* (*sol s*[*xfList*←*uInput*] *t*)
  proof(*cases t*=*0*)
  case *True*
    have (*sol s*[*xfList*←*uInput*] *0*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *0*)
    using *assms* **and** *to-sol-zero-its-dvars* **by** *blast*
    then show *?thesis* **using** *True* **by** *blast*
  next
    case *False*
    from *this* **have** *t* > *0* **using** *tHyp* **by** *simp*
    hence (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *vderiv-of* (λ *r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t*
    using *xfuHyp assms to-sol-greater-than-zero-its-dvars* **by** *blast*
   also have *vderiv-of* (λ*r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t* = *f* (*sol s*[*xfList*←*uInput*] *t*)
    using *assms xfuHyp* ‹*t* > *0*› **and** *vderiv-of-to-sol-its-vars* **by** *blast*
    ultimately show *?thesis* **by** *simp*
  qed
qed

lemma *conds4vdiffs*:
assumes *funcsHyp*:∀ *s g*. ∀ *xf*∈*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:∀ *uxf* ∈ *set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:∀ *t*≥*0*. ∀ *xf* ∈ *set xfList*. ((λτ. (*sol s*[*xfList*←*uInput*] τ) (π_1 *xf*))
has-vderiv-on (λτ. (π_2 *xf*) (*sol s*[*xfList*←*uInput*] τ))) {*0*..*t*}
shows ∀ *t* ≥ *0*. ∀ *xf* ∈ *set xfList*. (*sol s*[*xfList*←*uInput*] *t*) (∂ (π_1 *xf*)) = (π_2 *xf*) (*sol s*[*xfList*←*uInput*] *t*)
apply(*rule allI*, *rule impI*, *rule ballI*, *rule conds4vdiffs-prelim*)
using *assms* **by** *simp-all*

lemma *conds4Consts*:
assumes *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
shows ∀ *x*. *x* ∉ (π_1⦇*set xfList*⦈) ⟶ (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *0*
using *varsHyp* **apply**(*induct xfList uInput rule*: *list-induct2′*)
apply(*simp-all add*: *override-on-def varDiffs-def vdiff-def*)
by *clarsimp*

lemma *conds4InitState*:

assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *xfHyp*:(x, f) $\in$ *set xfList*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) x = s x
proof−
from *xfHyp* **obtain** u **where** *uxfHyp*:(u, x, f) $\in$ *set* (*uInput* $\otimes$ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
from *varsHyp* **have** *toZeroHyp*:(*sol s*) x = s x **using** *override-on-def xfHyp* **by** *auto*
from *uxfHyp* **and** *solHyp1* **have** u *0* (*sol s*) = (*sol s*) x **by** *fastforce*
also have (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) x = u *0* (*sol s*)
using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*
ultimately show (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) x = s x **using** *toZeroHyp* **by** *simp*
qed

lemma *conds4RestOfStrings*:
assumes $x \notin$ (π_1(|*set xfList*|)) $\cup$ *varDiffs*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] t) x = s x
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*)
by(*auto simp*: *varDiffs-def*)

lemma *conds4storeIVP-on-toSol*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf* *s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ $t \geq 0$. $\forall$ *xf* $\in$ *set xfList*.
((λt. (*sol s*[*xfList*$\leftarrow$*uInput*] t) (π_1 *xf*)) *has-vderiv-on* (λt. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] t))) {*0*..*t*}
shows *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*$\leftarrow$*uInput*] t)) *xfList s*
apply(*rule solves-store-ivpI*)
subgoal using *conds4vdiffs assms* **by** *blast*
subgoal using *conds4RestOfStrings* **by** *blast*
subgoal using *conds4Consts varsHyp* **by** *blast*
subgoal apply(*rule allI*, *rule impI*, *rule ballI*, *rule solves-odeI*)
  using *solHyp2* **by** *simp-all*
subgoal using *conds4InitState* **and** *assms* **by** *force*
done

theorem *dSolve-toSolve*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf* *s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)

and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *s*.$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *s*.$\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*.
((λ*t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λ*t*. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0*..*t*}
and *uniqHyp*:$\forall$ *s*.$\forall$ *X*. *solvesStoreIVP X xfList s* $\longrightarrow$ ($\forall$ *t* $\geq$ *0*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = *X t*)
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolve*)
subgoal using *assms* **and** *conds4storeIVP-on-toSol* **by** *simp*
subgoal by (*simp add*: *uniqHyp*)
using *postCondHyp postCondHyp* **by** *simp*

— As before, we keep refining the rule dSolve. This time we find the necessary restrictions to attain uniqueness.

lemma *conds4UniqSol*:
fixes *f*::*real store* $\Rightarrow$ *real*
assumes *tHyp*:*t* $\geq$ *0*
and *contHyp*:*continuous-on* ({*0*..*t*} $\times$ *UNIV*) (λ(*t*, (*r*::*real*)). *f* (φ_s *t*))
shows *unique-on-bounded-closed 0* {*0*..*t*} τ (λ*t r*. *f* (φ_s *t*)) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*))
apply(*simp add*: *ubc-definitions*, *rule conjI*)
subgoal using *contHyp continuous-rhs-def* **by** *fastforce*
subgoal using *assms continuous-rhs-def* **by** *fastforce*
done

lemma *solves-store-ivp-at-beginning-overrides*:
assumes *solvesStoreIVP* φ_s *xfList a*
shows φ_s *0* = *override-on a* (φ_s *0*) *varDiffs*
apply(*rule ext*, *subgoal-tac x* $\notin$ *varDiffs* $\longrightarrow$ φ_s *0 x* = *a x*)
subgoal by (*simp add*: *override-on-def*)
using *assms* **and** *solves-store-ivpD*(*6*) **by** *simp*

lemma *ubcStoreUniqueSol*:
assumes *tHyp*:*t* $\geq$ *0*
assumes *contHyp*:$\forall$ *xf* $\in$ *set xfList*. *continuous-on* ({*0*..*t*} $\times$ *UNIV*) (λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
and *eqDerivs*:$\forall$ *xf* $\in$ *set xfList*. $\forall$ τ $\in$ {*0*..*t*}. (π_2 *xf*) (φ_s τ) = (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ)
and *Fsolves*:*solvesStoreIVP* φ_s *xfList s*
and *solHyp*:*solvesStoreIVP* (λ τ. (*sol s*[*xfList*$\leftarrow$*uInput*] τ)) *xfList s*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = φ_s *t*
proof
fix *x*::*string* **show** (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = φ_s *t x*
proof(*cases x* $\in$ (π_1(|*set xfList*|)) $\cup$ *varDiffs*)

case *False*
then have *notInVars*:$x \notin (\pi_1 ⦇ set\ xfList ⦈) \cup varDiffs$ **by** *simp*
from *solHyp* **have** $(sol\ s[xfList \leftarrow uInput]\ t)\ x = s\ x$
using *tHyp notInVars solves-store-ivpD(1)* **by** *blast*
also from *Fsolves* **have** $\varphi_s\ t\ x = s\ x$ **using** *tHyp notInVars solves-store-ivpD(1)* **by** *blast*
ultimately show $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x$ **by** *simp*
next case *True*
then have $x \in (\pi_1 ⦇ set\ xfList ⦈) \vee x \in varDiffs$ **by** *simp*
from *this* **show** *?thesis*
proof
assume $x \in (\pi_1 ⦇ set\ xfList ⦈)$
from *this* **obtain** f **where** *xfHyp*:$(x, f) \in set\ xfList$ **by** *fastforce*

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

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

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

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

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

apply(*clarify*) **apply**(*rule conds4UniqSol*) **by**(*auto*)
from *this* **have** *3*:$unique\text{-}on\text{-}bounded\text{-}closed\ 0\ \{0..t\}\ (s\ x)\ (\lambda\tau\ r.\ f\ (sol\ s[xfList \leftarrow uInput]\ \tau))$
$UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$ **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 $x \in varDiffs$

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

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

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

theorem $dSolve\text{-}toSolveUBC$:
assumes $funcsHyp{:}\forall\ s\ g.\ \forall\ xf{\in}set\ xfList.\ \pi_2\ xf\ (override\text{-}on\ s\ g\ varDiffs) = \pi_2\ xf$
s
and $distinctHyp{:}distinct\ (map\ \pi_1\ xfList)$
and $lengthHyp{:}length\ xfList = length\ uInput$

and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *s*. $\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). π_1 *uxf 0* (*sol s*) = *sol s* (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *s*. $\forall$ *t*$\geq$*0*. $\forall$ *xf*$\in$*set xfList*. ((λ*t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on*
(λ*t*. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0..t*}
and *contHyp*:$\forall$ *s*. $\forall$ *t*$\geq$*0*. $\forall$ *xf* $\in$ *set xfList*. *continuous-on* ({*0..t*} $\times$ *UNIV*)
(λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
and *uniqHyp*:$\forall$ *s*. $\forall$ φ_s. φ_s *solvesTheStoreIVP xfList withInitState s* $\longrightarrow$
($\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*. $\forall$ *r* $\in$ {*0..t*}. (π_2 *xf*) (φ_s *r*) = (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *r*))
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput=uInput* **in** *dSolveUBC*)
using *contHyp* **apply** *simp*
apply(*rule allI*, *rule-tac uInput=uInput* **in** *conds4storeIVP-on-toSol*)
using *assms* **by** *auto*

1.4.4 "Differential Invariant."

lemma *solvesStoreIVP-couldBeModified*:
fixes *F*::*real* $\Rightarrow$ *real store*
assumes *vars*:$\forall$ *t*$\geq$*0*. $\forall$ *xf*$\in$*set xfList*. ((λ*t*. *F t* (π_1 *xf*)) *solves-ode* (λ*t r*. π_2 *xf* (*F t*))) {*0..t*} *UNIV*
and *dvars*:$\forall$ *t* $\geq$ *0*. $\forall$ *xf*$\in$*set xfList*. (*F t* (∂ (π_1 *xf*))) = (π_2 *xf*) (*F t*)
shows $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0..t*}. $\forall$ *xf* $\in$ *set xfList*.
((λ *t*. *F t* (π_1 *xf*)) *has-vector-derivative F r* (∂ (π_1 *xf*))) (*at r within* {*0..t*})
proof(*clarify*, *rename-tac t r x f*)
fix *x f* **and** *t r*::*real*
assume *tHyp*:*0* $\leq$ *t* **and** *xfHyp*:(*x*, *f*) $\in$ *set xfList* **and** *rHyp*:*r* $\in$ {*0..t*}
from *this* **and** *vars* **have** ((λ*t*. *F t x*) *solves-ode* (λ*t r*. *f* (*F t*))) {*0..t*} *UNIV*
using *tHyp* **by** *fastforce*
hence $*$:$\forall$ *r*$\in${*0..t*}. ((λ *t*. *F t x*) *has-vector-derivative* (λ *t*. *f* (*F t*)) *r*) (*at r within* {*0..t*})
by (*simp add*: *solves-ode-def has-vderiv-on-def tHyp*)
have $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0..t*}. $\forall$ *xf* $\in$ *set xfList*. (*F r* (∂ (π_1 *xf*))) = (π_2 *xf*) (*F r*)
using *assms* **by** *auto*
from *this rHyp* **and** *xfHyp* **have** (*F r* (∂ *x*)) = *f* (*F r*) **by** *force*
then show ((λ*t*. *F t* (π_1 (*x*, *f*))) *has-vector-derivative F r* (∂ (π_1 (*x*, *f*)))) (*at r within* {*0..t*})
using $*$ *rHyp* **by** *auto*
qed

lemma *derivationLemma-baseCase*:
fixes *F*::*real* $\Rightarrow$ *real store*
assumes *solves*:*solvesStoreIVP F xfList a*
shows $\forall$ *x* $\in$ (*UNIV* $-$ *varDiffs*). $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0..t*}.
((λ *t*. *F t x*) *has-vector-derivative F r* (∂ *x*)) (*at r within* {*0..t*})
proof

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

lemma *derivationLemma*:
assumes *solvesStoreIVP F xfList a*
and *tHyp*:$t \geq 0$
and *termVarsHyp*:$\forall\ x \in trmVars\ \eta.\ x \in (UNIV - varDiffs)$
shows $\forall\, r{\in}\{0..t\}.\ ((\lambda\ s.\ [\![\eta]\!]_t\ (F\ s))$*has-vector-derivative* $[\![\partial_t\ \eta]\!]_t\ (F\ r))\ (at\ r\ within\ \{0..t\})$
using *termVarsHyp* **proof**(*induction* η)
case (*Const r*)
then show *?case* **by** *simp*
next
case (*Var y*)
then have *yHyp*:$y \in UNIV - varDiffs$ **by** *auto*
from *this tHyp* **and** *assms*(*1*) **show** *?case*
using *derivationLemma-baseCase* **by** *auto*

next
  case (*Mns* η)
  then show *?case*
  apply(*clarsimp*)
  by(*rule derivative-intros*, *simp*)
next
  case (*Sum* η*1* η*2*)
  then show *?case*
  apply(*clarsimp*)
  by(*rule derivative-intros*, *simp-all*)
next
  case (*Mult* η*1* η*2*)
  then show *?case*
  apply(*clarsimp*)
  apply(*subgoal-tac* ((λs. $[\![\eta 1]\!]_t$ $(F\ s)$ $*_R$ $[\![\eta 2]\!]_t$ $(F\ s)$) *has-vector-derivative* $[\![\partial_t\ \eta 1]\!]_t$ $(F\ r)$ $\cdot$ $[\![\eta 2]\!]_t$ $(F\ r)$ $+$ $[\![\eta 1]\!]_t$ $(F\ r)$ $\cdot$ $[\![\partial_t\ \eta 2]\!]_t$ $(F\ r)$) (*at r within* $\{0..t\}$),*simp*)
  apply(*rule-tac* $f'1$=$[\![\partial_t\ \eta 1]\!]_t$ $(F\ r)$ **and** $g'1$=$[\![\partial_t\ \eta 2]\!]_t$ $(F\ r)$ **in** *derivative-eq-intros*(*25*))
  by (*simp-all add*: *has-field-derivative-iff-has-vector-derivative*)
qed

lemma *diff-subst-prprty-4terms*:
assumes *solves*:$\forall$ *xf* $\in$ *set xfList*. $F\ t$ $(\partial\ (\pi_1\ xf))$ $=$ $\pi_2\ xf$ $(F\ t)$
and *tHyp*:(*t*::*real*) $\geq$ *0*
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
and *termVarsHyp*:*trmVars* η $\subseteq$ (*UNIV* $-$ *varDiffs*)
shows $[\![\partial_t\ \eta]\!]_t$ $(F\ t)$ $=$ $[\![$((*map* (*vdiff* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)$\langle\partial_t\ \eta\rangle]\!]_t$ $(F\ t)$
using *termVarsHyp* **apply**(*induction* η) **apply**(*simp-all add*: *substList-help2*)
using *listsHyp* **and** *solves* **apply**(*induct xfList uInput rule*: *list-induct2'*, *simp*, *simp*, *simp*)
proof(*clarify*, *rename-tac y g xfTail* ϑ *trmTail x*)
fix *x y*::*string* **and** ϑ::*trms* **and** *g* **and** *xfTail*::((*string* $\times$ (*real store* $\Rightarrow$ *real*)) *list*)
and *trmTail*
assume *IH*:$\bigwedge x$. $x \notin$ *varDiffs* $\Longrightarrow$ *map* π_2 *xfTail* = *map tval trmTail* $\Longrightarrow$
$\forall$ *xf*$\in$*set xfTail*. $F\ t$ $(\partial\ (\pi_1\ xf))$ $=$ $\pi_2\ xf$ $(F\ t)$ $\Longrightarrow$
$F\ t$ $(\partial\ x)$ $=$ $[\![$(*map* (*vdiff* $\circ$ π_1) *xfTail* $\otimes$ *trmTail*)$\langle t_V\ (\partial\ x)\rangle]\!]_t$ $(F\ t)$
and *1*:$x \notin$ *varDiffs* **and** *2*:*map* π_2 ((*y*, *g*) # *xfTail*) = *map tval* (ϑ # *trmTail*)
and *3*:$\forall$ *xf*$\in$*set* ((*y*, *g*) # *xfTail*). $F\ t$ $(\partial\ (\pi_1\ xf))$ $=$ $\pi_2\ xf$ $(F\ t)$
hence $*$:$[\![$(*map* (*vdiff* $\circ$ π_1) *xfTail* $\otimes$ *trmTail*)$\langle$*Var* $(\partial\ x)\rangle]\!]_t$ $(F\ t)$ $=$ $F\ t$ $(\partial\ x)$
using *tHyp* **by** *auto*
show $F\ t$ $(\partial\ x)$ $=$ $[\![$((*map* (*vdiff* $\circ$ π_1) ((*y*, *g*) # *xfTail*)) $\otimes$ (ϑ # *trmTail*)) $\langle t_V$ $(\partial\ x)\rangle]\!]_t$ $(F\ t)$
  proof(*cases* $x \in$ *set* (*map* π_1 ((*y*, *g*) # *xfTail*)))
    case *True*
    then have $x = y \lor (x \neq y \land x \in$ *set* (*map* π_1 *xfTail*)) **by** *auto*
    moreover
    {**assume** $x = y$
      from *this* **have** ((*map* (*vdiff* $\circ$ π_1) ((*y*, *g*) # *xfTail*)) $\otimes$ (ϑ # *trmTail*))$\langle t_V$ $(\partial\ x)\rangle$ $=$ ϑ **by** *simp*

also from *3 tHyp* **have** $F\ t\ (\partial\ y) = g\ (F\ t)$ **by** *simp*
moreover from *2* **have** $[\![\vartheta]\!]_t\ (F\ t) = g\ (F\ t)$ **by** *simp*
ultimately have *?thesis* **by** (*simp add*: ‹$x = y$›)**}**
moreover
{assume $x \neq y \wedge x \in set\ (map\ \pi_1\ xfTail)$
then have $\partial\ x \neq \partial\ y$ **using** *vdiff-inj* **by** *auto*
from *this* **have** $((map\ (vdiff \circ \pi_1)\ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V\ (\partial\ x)\rangle =$
$((map\ (vdiff \circ \pi_1)\ xfTail) \otimes trmTail)\ \langle t_V\ (\partial\ x)\rangle$ **by** *simp*
hence *?thesis* **using** $*$ **by** *simp***}**
ultimately show *?thesis* **by** *blast*
next
case *False*
then have $((map\ (vdiff \circ \pi_1)\ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V\ (\partial\ x)\rangle = t_V\ (\partial\ x)$
using *substList-cross-vdiff-on-non-ocurring-var* **by**(*metis*(*no-types*, *lifting*) *List.map.compositionality*)
thus *?thesis* **by** *simp*
qed
qed

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

lemma *non-empty-funList-implies-non-empty-trmList*:
shows $\forall\ list.(x,f) \in set\ list \wedge map\ \pi_2\ list = map\ tval\ tList \longrightarrow (\exists\ \vartheta.[\![\vartheta]\!]_t = f \wedge \vartheta \in set\ tList)$
by(*induction tList*, *auto*)

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

using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$ **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) = 0$ **using** *aHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative*

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

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

lemma *diff-subst-prprty-4props*:
assumes *solves*:$\forall\ xf \in set\ xfList.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$
and *tHyp*:$t \geq 0$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *propVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
shows $[\![\partial_P\ \varphi]\!]_P\ (F\ t) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\upharpoonleft\partial_P\ \varphi\upharpoonright]\!]_P\ (F\ t)$
using *propVarsHyp* **apply**(*induction* φ, *simp-all*)
using *assms diff-subst-prprty-4terms* **apply** *fastforce*
using *assms diff-subst-prprty-4terms* **apply** *fastforce*
using *assms diff-subst-prprty-4terms* **by** *fastforce*

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

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

hence $\exists r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![(\partial_t\ \eta)]\!]_t)\ (F\ r)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain *r* **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \land 0 \leq r \land r \leq t \land [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\land [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c > 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(*45*)
not-le)
next
show $0 \leq [\![\eta]\!]_t\ a \longrightarrow (\forall c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow 0 \leq [\![\eta]\!]_t\ c)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a \geq 0$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t \geq 0 \land F\ t = c \land solvesStoreIVP\ F\ xfList\ a \land (\forall r \in \{0..t\}.\ G\ (F\ r))$

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

hence $\exists r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain r **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\wedge\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c \geq 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(*45*)
not-le)
qed
qed

lemma *less-pval-to-tval*:
assumes $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ (\vartheta \prec \eta) \restriction]\!]_P\ st$
shows $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t\ (\eta \oplus (\ominus\ \vartheta)) \rangle]\!]_t\ st \geq 0$
using *assms* **by**(*auto*)

lemma *leq-pval-to-tval*:
assumes $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ (\vartheta \preceq \eta) \restriction]\!]_P\ st$
shows $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t\ (\eta \oplus (\ominus\ \vartheta)) \rangle]\!]_t\ st \geq 0$
using *assms* **by**(*auto*)

lemma *dInv-prelim*:
assumes *substHyp*:$\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) =$
$0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ \varphi \restriction]\!]_P\ st$
and *propVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$

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

ultimately show *?case* **by** *simp*
next
case $(And\ \varphi 1\ \varphi 2)$
then show *?case* **by**(*simp*)
next
case $(Or\ \varphi 1\ \varphi 2)$
from *this* **show** *?case* **by** *auto*
qed
qed

theorem *dInv*:
assumes $\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!| set\ xfList |\!))) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff\ \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P\ \varphi \restriction]\!]_P\ st$
and $termVarsHyp{:}propVars\ \varphi \subseteq (UNIV\ -\ varDiffs)$
and $listsHyp{:}map\ \pi_2\ xfList = map\ tval\ uInput$
and $phi\text{-}p{:}P = [\![\varphi]\!]_P$
shows $PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ P$
proof(*clarsimp*)
fix *a b*
assume $(a,\ b) \in \lceil P \rceil$
from *this* **have** $aHyp{:}a = b \wedge P\ a$ **by** (*metis* (*full-types*) *d-p2r rdom-p2r-contents*)
have $P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c)$
using *assms dInv-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c$ **by** *blast*
thus $(a,\ b) \in wp\ (ODEsystem\ xfList\ with\ G\ )\ \lceil P \rceil$
using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)
qed

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

end

theory *VC-diffKAD-examples*
imports *VC-diffKAD*

begin

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*:
$\quad PRE\ (\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' > 0)$
$\quad (ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v''))]\ with\ (\lambda\ s.\ True))$
$\quad POST\ (\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac uInput*=[$\lambda\ t\ s.\ s\ ''v'' \cdot t + s\ ''x''$] **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal by**(*simp add*: *wp-trafo vdiff-def add-strict-increasing2*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
prefer *2* **apply**(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*)
apply(*clarify*, *rule-tac* $f'1=\lambda\ x.\ s\ ''v''$ **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*:φ_s *solvesTheStoreIVP* $[(x,\ \lambda s.\ s\ v),\ (v,\ \lambda s.\ s\ a)]$ *withInitState s*
and *tHyp*:$r \leq t$ **and** *rHyp*:$0 \leq r$ **and** *distinct*:$x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin$ *varDiffs*
shows $\varphi_s\ r\ v = s\ a \cdot r + s\ v$
proof−
from *assms* **have** *1*:$((\lambda t.\ \varphi_s\ t\ v)$ *solves-ode* $(\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}\ UNIV \wedge \varphi_s\ 0\ v = s\ v$
by (*simp add*: *solvesStoreIVP-def*)
from *assms* **have** *obs*:$\forall\ r \in \{0..t\}.\ \varphi_s\ r\ a = s\ a$
by(*auto simp*: *solvesStoreIVP-def varDiffs-def*)
have *2*:$((\lambda t.\ s\ a \cdot t + s\ v)$ *solves-ode* $(\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}\ UNIV$
unfolding *solves-ode-def* **apply**(*subgoal-tac* $((\lambda x.\ s\ a \cdot x + s\ v)$ *has-vderiv-on* $(\lambda x.\ s\ a))\ \{0..t\})$
using *obs* **apply** (*simp add*: *has-vderiv-on-def*) **by**(*rule galilean-transform*)
have *3*:*unique-on-bounded-closed 0* $\{0..t\}\ (s\ v)\ (\lambda t\ r.\ \varphi_s\ t\ a)\ UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$
apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*
apply(*rule continuous-intros*)
apply(*auto intro*: *continuous-intros*)
by (*metis continuous-on-const continuous-on-eq*)

thus φ_s *r v* = *s a* · *r* + *s v*
  apply(*rule-tac unique-on-bounded-closed.unique-solution*[*of 0* {*0..t*} *s v*
  (λ*t r*. φ_s *t a*) *UNIV* (*if t* = *0 then 1 else 1* / (*t* + *1*)) (λ*t*. φ_s *t v*)])
  using *rHyp tHyp 1 2* **and** *3* **by** *auto*
qed

lemma *motion-with-constant-acceleration*:
    PRE (λ *s*. *s* ''*y*'' < *s* ''*x*'' $\wedge$ *s* ''*v*'' $\geq$ *0* $\wedge$ *s* ''*a*'' > *0*)
    (*ODEsystem* [(''*x*'',(λ *s*. *s* ''*v*'')),(''*v*'',(λ *s*. *s* ''*a*''))] *with* (λ *s*. *True*))
    POST (λ *s*. (*s* ''*y*'' < *s* ''*x*''))
apply(*rule-tac uInput*=[λ *t s*. *s* ''*a*'' · *t* ˆ *2*/*2* + *s* ''*v*'' · *t* + *s* ''*x*'',
 λ *t s*. *s* ''*a*'' · *t* + *s* ''*v*''] **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal** **by**(*simp add*: *wp-trafo vdiff-def add-strict-increasing2*)
prefer *6* **subgoal**
  apply(*simp add*: *vdiff-def*, *clarify*, *rule conjI*)
  by(*rule galilean-transform*)+
prefer *6* **subgoal**
  apply(*simp add*: *vdiff-def*, *safe*)
  by(*rule continuous-intros*)+
prefer *6* **subgoal**
  apply(*simp add*: *vdiff-def*, *safe*)
  subgoal for *s* φ_s *t r* **apply**(*rule flow-vel-is-galilean-vel*[*of* φ_s ''*x*'' - - - - *t*])
   by(*simp-all add*: *varDiffs-def vdiff-def*)
  apply(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*) **done**
by(*auto simp*: *varDiffs-def vdiff-def*)

Example of a hybrid system with two modes verified with the equality dS. We also need to provide a previous (similar) lemma.

lemma *flow-vel-is-galilean-vel2*:
assumes *solHyp*:φ_s *solvesTheStoreIVP* [(*x*, λ*s*. *s v*), (*v*, λ*s*. − *s a*)] *withInitState* *s*
  and *tHyp*:*r* $\leq$ *t* **and** *rHyp*:*0* $\leq$ *r* **and** *distinct*:*x* $\neq$ *v* $\wedge$ *v* $\neq$ *a* $\wedge$ *x* $\neq$ *a* $\wedge$ *a* $\notin$ *varDiffs*
shows φ_s *r v* = *s v* − *s a* · *r*
proof−
from *assms* **have** *1*:((*λt*. φ_s *t v*) *solves-ode* (λ*t r*. − φ_s *t a*)) {*0..t*} *UNIV* $\wedge$ φ_s *0 v* = *s v*
 by (*simp add*: *solvesStoreIVP-def*)
from *assms* **have** *obs*:$\forall$ *r* $\in$ {*0..t*}. φ_s *r a* = *s a*
 by(*auto simp*: *solvesStoreIVP-def varDiffs-def*)
have *2*:((λ*t*. − *s a* · *t* + *s v*) *solves-ode* (λ*t r*. − φ_s *t a*)) {*0..t*} *UNIV*
 unfolding *solves-ode-def* **apply**(*subgoal-tac* ((λ*x*. − *s a* · *x* + *s v*) *has-vderiv-on* (λ*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*) (λ*t r*. − φ_s *t a*) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*))
  apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
  using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
  apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*

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

lemma *single-hop-ball*:
PRE (λ *s. 0* ≤ *s* ''*x*'' ∧ *s* ''*x*'' = *H* ∧ *s* ''*v*'' = *0* ∧ *s* ''*g*'' > *0* ∧ *1* ≥ *c* ∧ *c* ≥ *0*)
(((*ODEsystem* [(''*x*'', λ *s. s* ''*v*''),(''*v*'',λ *s.* − *s* ''*g*'')] *with* (λ *s. 0* ≤ *s* ''*x*''))));
(*IF* (λ *s. s* ''*x*'' = *0*) *THEN* (''*v*'' ::= (λ *s.* − *c* · *s* ''*v*'')) *ELSE* (''*v*'' ::= (λ *s. s* ''*v*'')) *FI*))
POST (λ *s. 0* ≤ *s* ''*x*'' ∧ *s* ''*x*'' ≤ *H*)
apply(*simp*, *subst dS*[*of* [λ *t s.* − *s* ''*g*'' · *t* ˆ *2/2* + *s* ''*v*'' · *t* + *s* ''*x*'', λ *t s.* − *s* ''*g*'' · *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 τ*
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 (λ *s. s* ''*x*'' = *0*)
(*ODEsystem* [(''*x*'',(λ *s. s* ''*x*'' + *1*))] *with* (λ *s. s* ''*x*'' ≥ *0*))
POST (λ *s. s* ''*x*'' ≥ *0*)
using *dWeakening* **by** *blast*

lemma *system-where-the-guard-implies-the-postcondition2*:
PRE (λ *s. s* ''*x*'' = *0*)
(*ODEsystem* [(''*x*'',(λ *s. s* ''*x*'' + *1*))] *with* (λ *s. s* ''*x*'' ≥ *0*))
POST (λ *s. s* ''*x*'' ≥ *0*)

apply(*clarify*, *simp add*: *p2r-def*)
apply(*simp add*: *rel-ad-def rel-antidomain-kleene-algebra.addual.ars-r-def*)
apply(*simp add*: *rel-antidomain-kleene-algebra.fbox-def*)
apply(*simp add*: *relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def*)
by *auto*

— Example of system proved with a differential invariant.
lemma *circular-motion*:
$PRE\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''y'')),(''y'',(\lambda\ s.\ -\ s\ ''x''))]\ with\ G)$
$POST\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
apply(*rule-tac* η=$(t_V\ ''x'')\odot(t_V\ ''x'') \oplus (t_V\ ''y'')\odot(t_V\ ''y'') \oplus (\ominus(t_V\ ''r'')\odot(t_V\ ''r''))$
and *uInput*=$[t_V\ ''y'', \ominus\ (t_V\ ''x'')]$ **in** *dInvForTrms*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*clarsimp*, *erule-tac* $x=''r''$ **in** *allE*)
by *simp*

— Example of systems proved with differential invariants, cuts and weakenings.
declare *d-p2r* [*simp del*]
lemma *motion-with-constant-velocity-and-invariants*:
$PRE\ (\lambda\ s.\ s\ ''x'' > s\ ''y'' \wedge s\ ''v'' > 0)$
$(ODEsystem\ [(''x'', \lambda\ s.\ s\ ''v'')]\ with\ (\lambda\ s.\ True))$
$POST\ (\lambda\ s.\ s\ ''x''>\ s\ ''y'')$
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''v'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \prec (t_V\ ''v'')$ **and** *uInput*=$[t_V\ ''v'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''v''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' > s\ ''y''$ **in** *dCut*)
apply(*rule-tac* φ=$(t_V\ ''y'') \prec (t_V\ ''x'')$ **and** *uInput*=$[t_V\ ''v'']$ **and**
F=$\lambda\ s.\ \ s\ ''x'' > s\ ''y''$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''y''$ **in** *allE*, *simp*)
using *dWeakening* **by** *simp*

lemma *motion-with-constant-acceleration-and-invariants*:
$PRE\ (\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]\ with\ (\lambda\ s.\ True))$
$POST\ (\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''a'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \prec (t_V\ ''a'')$ **and** *uInput*=$[t_V\ ''v'', t_V\ ''a'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''a''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''v'' \geq 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \preceq (t_V\ ''v'')$ **and** *uInput*=$[t_V\ ''v'', t_V\ ''a'']$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' >\ \ s\ ''y''$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ ''y'') \prec (t_V\ ''x'')$ **and** *uInput*=$[t_V\ ''v'', t_V\ ''a'']$**in** *dInvFinal*)
apply(*simp-all add*: *varDiffs-def vdiff-def*, *clarify*, *erule-tac* $x=''y''$ **in** *allE*, *simp*)
using *dWeakening* **by** *simp*

— We revisit the two modes example from before, and prove it with invariants.

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

— Finally, we add a well known example in the hybrid systems community, the bouncing ball.

lemma *bouncing-ball-invariant*:$0 \leq x \Longrightarrow 0 < g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \Longrightarrow (x{::}real) \leq H$
proof−
assume $0 \leq x$ **and** $0 < g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v$
then have $v \cdot v = 2 \cdot g \cdot H - 2 \cdot g \cdot x \wedge 0 < g$ **by** *auto*
hence $*$:$v \cdot v = 2 \cdot g \cdot (H - x) \wedge 0 < g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
from *this* **have** $(v \cdot v)/(2 \cdot g) = (H - x)$ **by** *auto*
also from $*$ **have** $(v \cdot v)/(2 \cdot g) \geq 0$
by (*meson divide-nonneg-pos linordered-field-class.sign-simps*(*44*) *zero-less-numeral*)

ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
$PRE\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0)$

$((ODEsystem\ [(''x'', \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s. - s\ ''g'')]\ with\ (\lambda\ s.\ 0 \leq s\ ''x''));$
$(IF\ (\lambda\ s.\ s\ ''x'' = 0)\ THEN\ (''v'' ::= (\lambda\ s. - s\ ''v''))\ ELSE\ (Id)\ FI))^*$
$POST\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*rule rel-antidomain-kleene-algebra.fbox-starI*[*of* - $\lceil\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge$
$2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - (s\ ''v'' \cdot s\ ''v'')\rceil$])
apply(*simp*, *simp add*: *d-p2r*)
apply(*subgoal-tac*
$rdom\ \lceil\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''\rceil$
$\subseteq wp\ (ODEsystem\ [(''x'', \lambda s.\ s\ ''v''), (''v'', \lambda s. - s\ ''g'')]\ with\ (\lambda s.\ 0 \leq s\ ''x''))$
$\lceil inf\ (sup\ (-\ (\lambda s.\ s\ ''x'' = 0))\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' =$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))$
$(sup\ (\lambda s.\ s\ ''x'' = 0)\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' =$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))\rceil$)
apply(*simp add*: *d-p2r*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = ((t_C\ \ 0) \prec (t_V\ \ ''g''))$ **and** *uInput*=[$t_V\ \ ''v''$, $\ominus\ t_V\ \ ''g''$]**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''g''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ 2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 2) \odot (t_V\ \ ''g'') \odot (t_C\ H) \oplus (\ominus\ ((t_V\ \ ''v'') \odot (t_V\ \ ''v'')))$
$\doteq (t_C\ 2) \odot (t_V\ \ ''g'') \odot (t_V\ \ ''x'')$ **and** *uInput*=[$t_V\ \ ''v''$, $\ominus\ t_V\ \ ''g''$]**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''g''$ **in** *allE*, *simp*)
apply(*rule dWeakening*, *clarsimp*)
using *bouncing-ball-invariant* **by** *auto*

declare *d-p2r* [*simp*]

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

begin

2 Hybrid Systems Preliminaries

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

2.1 Real Numbers

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

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

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

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

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

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

lemma *sin-cos-squared-add3*:$(x::('a::\ \{banach,real\text{-}normed\text{-}field\})) * (sin\ t)^2 + x * (cos\ t)^2 = x$
  by(*subst semiring-factor-left*, *subst sin-cos-squared-add*, *simp*)

lemma *sin-cos-squared-add4*:$(x::('a::\ \{banach,real\text{-}normed\text{-}field\})) * (cos\ t)^2 + x * (sin\ t)^2 = x$
  by(*subst semiring-factor-left*, *subst sin-cos-squared-add2*, *simp*)

lemma [*simp*]:$((x::real) * cos\ t - y * sin\ t)^2 + (x * sin\ t + y * cos\ t)^2 = x^2 + y^2$
proof−
  have $(x * cos\ t - y * sin\ t)^2 = x^2 * (cos\ t)^2 + y^2 * (sin\ t)^2 - 2 * (x * cos\ t) * (y * sin\ t)$
    by(*simp add*: *power2-diff power-mult-distrib*)
  also have $(x * sin\ t + y * cos\ t)^2 = y^2 * (cos\ t)^2 + x^2 * (sin\ t)^2 + 2 * (x * cos\ t) * (y * sin\ t)$
    by(*simp add*: *power2-sum power-mult-distrib*)
  ultimately show $(x * cos\ t - y * sin\ t)^2 + (x * sin\ t + y * cos\ t)^2 = x^2 + y^2$

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

qed

2.2 Unit vectors and vector norm

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

lemma *squared-norm-vec*:$(norm\ x)^2 = (\sum i{\in}UNIV.\ (x\ \$\ i)^2)$

unfolding *norm-vec-def L2-set-def* **by** (*simp add*: *sum-nonneg*)

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

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

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

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

2.3 Matrix norm

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

lemma *unit-norms-bound*:
fixes $A::real\hat{}('n::finite)\hat{}('m::finite)$
shows $norm\ x = 1 \Longrightarrow norm\ (A *v\ x) \leq norm\ ((\chi\ i\ j.\ |A\ \$\ i\ \$\ j|) *v\ 1)$
proof−
assume $norm\ x = 1$
from *this* **have** $\bigwedge j.\ |x\ \$\ j| \leq 1$
by (*metis component-le-norm-cart*)
then have $\bigwedge i\ j.\ |A\ \$\ i\ \$\ j| * |x\ \$\ j| \leq |A\ \$\ i\ \$\ j| * 1$
using *mult-left-mono* **by** (*simp add*: *mult-left-le*)
from *this* **have** $\bigwedge i.(\sum j\in UNIV.\ |A\ \$\ i\ \$\ j| * |x\ \$\ j|)^2 \leq (\sum j\in UNIV.\ |A\ \$\ i\ \$\ j|)^2$
by (*simp add*: *power-mono sum-mono sum-nonneg*)
also have $\bigwedge i.(\sum j\in UNIV.\ A\ \$\ i\ \$\ j * x\ \$\ j)^2 \leq (\sum j\in UNIV.\ |A\ \$\ i\ \$\ j * x\ \$\ j|)^2$
using *abs-le-square-iff* **by** *force*
moreover have $\bigwedge i.(\sum j\in UNIV.\ |A\ \$\ i\ \$\ j * x\ \$\ j|)^2 = (\sum j\in UNIV.\ |A\ \$\ i\ \$\ j| * |x\ \$\ j|)^2$
by (*simp add*: *abs-mult*)
ultimately have $\bigwedge i.(\sum j\in UNIV.\ A\ \$\ i\ \$\ j * x\ \$\ j)^2 \leq (\sum j\in UNIV.\ |A\ \$\ i\ \$\ j|)^2$

using *order-trans* **by** *fastforce*
hence $(\sum i \in UNIV. (\sum j \in UNIV. A \$ i \$ j * x \$ j)^2) \leq (\sum i \in UNIV. (\sum j \in UNIV. |A \$ i \$ j|)^2)$
by(*simp add*: *sum-mono*)
then have $(sqrt\ (\sum i \in UNIV. (\sum j \in UNIV. A \$ i \$ j * x \$ j)^2)) \leq (sqrt\ (\sum i \in UNIV. (\sum j \in UNIV. |A \$ i \$ j|)^2))$
using *real-sqrt-le-mono* **by** *blast*
thus $norm\ (A *v\ x) \leq norm\ ((\chi\ i\ j.\ |A \$ i \$ j|) *v\ 1)$
by(*simp add*: *norm-vec-def L2-set-def matrix-vector-mult-def*)
qed

lemma *unit-norms-exists*:
fixes $A::real\hat{}('n::finite)\hat{}('m::finite)$
shows *bounded*:*bounded* $\{norm\ (A *v\ x) \mid x.\ norm\ x = 1\}$
and *bdd-above*:*bdd-above* $\{norm\ (A *v\ x) \mid x.\ norm\ x = 1\}$
and *non-empty*:$\{norm\ (A *v\ x) \mid x.\ norm\ x = 1\} \neq \{\}$ (**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 $\bigwedge k::'n.\ norm\ (axis\ k\ (1::real)) = 1$
using *norm-axis-1* **by** *blast*
hence $\bigwedge k::'n.\ norm\ ((A::real\hat{}('n::finite)\hat{}'m) *v\ (axis\ k\ (1::real))) \in\ ?U$
by *blast*
thus $?U \neq \{\}$ **by** *blast*
qed

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

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

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

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

lemma *maxAbs-def*:*maxAbs* (*A*::*real*^'*n*^'*m*) = *Max* { |*A* $ *i* $ *j*| |*i j*. *i* ∈ (*UNIV*::'*m* *set*) ∧ *j* ∈ (*UNIV*::'*n* *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*| |*i j*. *i* ∈ (*UNIV*::'*m* *set*) ∧ *j* ∈ (*UNIV*::'*n* *set*)} (**is** *finite ?X*)
proof−
  {**fix** *i*::'*m*
    have *finite* {|*A* $ *i* $ *j*| | *j*. *j* ∈ (*UNIV*::'*n* *set*)}
      using *finite-Atleast-Atmost-nat* **by** *fastforce*}
  hence ∀ *i*::'*m*. *finite* {|*A* $ *i* $ *j*| | *j*. *j* ∈ (*UNIV*::'*n* *set*)} **by** *blast*
  then have *finite* (⋃*i*∈*UNIV*. {|*A* $ *i* $ *j*| | *j*. *j* ∈ (*UNIV*::'*n* *set*)}) (**is** *finite ?Y*)
    using *finite-class.finite-UNIV* **by** *blast*
  also have *?X* ⊆ *?Y* **by** *auto*
  ultimately show *?thesis* **using** *finite-subset* **by** *blast*
qed

lemma *maxAbs-ge-0*:*maxAbs A* ≥ *0*
proof−
  have ⋀ *i j*. |*A* $ *i* $ *j*| ≥ *0* **by** *simp*
  also have ⋀ *i j*. *maxAbs A* ≥ |*A* $ *i* $ *j*|
    unfolding *maxAbs-def* **using** *finite-matrix-abs Max-ge maxAbs-def* **by** *blast*
  finally show *0* ≤ *maxAbs A* **.**
qed

lemma *norms-le-dims-maxAbs*:
  fixes *A*::*real*^('*n*::*finite*)^('*m*::*finite*)
  shows $norm_S$ *A* ≤ *real CARD*('*n*) ∗ *real CARD*('*m*) ∗(*maxAbs A*) (**is** $norm_S$ *A* ≤*?n* ∗ *?m* ∗ (*maxAbs A*))
proof−
  {**fix** *x*::(*real*, '*n*) *vec* **assume** *norm x* = *1*
    hence *comp-le-1*:∀ *i*::'*n*. |*x* $ *i*| ≤ *1*
      by (*simp add*: *norm-bound-component-le-cart*)
    have *A* ∗*v x* = (∑*i*∈*UNIV*. *x* $ *i* ∗*s column i A*)
      using *matrix-mult-sum* **by** *blast*
    hence *norm* (*A* ∗*v x*) ≤ (∑(*i*::'*n*)∈*UNIV*. *norm* (*x* $ *i* ∗*s column i A*))
      by (*simp add*: *sum-norm-le*)
    also have ... = (∑(*i*::'*n*)∈*UNIV*. |*x* $ *i*| ∗ *norm* (*column i A*))
      by (*simp add*: *norm-scalar-mult*)
    also have ... ≤ (∑(*i*::'*n*)∈*UNIV*. *norm* (*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 ... ≤ (∑(*i*::'*n*)∈*UNIV*. *?m* ∗ *maxAbs A*)
    proof(*unfold norm-vec-def L2-set-def real-norm-def*)

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

2.4 Derivatives

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

lemma *convergences-solves-vec-nth*:
assumes $((\lambda y.\ (\varphi\ y - \varphi\ (netlimit\ (at\ x\ within\ \{0..t\})) - (y - netlimit\ (at\ x\ within\ \{0..t\})) *_R f\ (\varphi\ x))\ /_R$
$|y - netlimit\ (at\ x\ within\ \{0..t\})|) \longrightarrow 0)\ (at\ x\ within\ \{0..t\})$ (**is** $((\lambda y.\ ?f\ y) \longrightarrow 0)\ ?net$)
shows $((\lambda y.\ (\varphi\ y\ \$\ i - \varphi\ (netlimit\ (at\ x\ within\ \{0..t\}))\ \$\ i - (y - netlimit\ (at\ x\ within\ \{0..t\})) *_R f\ (\varphi\ x)\ \$\ i)\ /_R$
$|y - netlimit\ (at\ x\ within\ \{0..t\})|) \longrightarrow 0)\ (at\ x\ within\ \{0..t\})$ (**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*::*banach*) ^('*n*::*finite*)) ⇒ ('*a* ^'*n*)
assumes $(\varphi$ *solves-ode* $(\lambda\ t.\ f))\ \{0..t\}$ *UNIV*
shows $((\lambda\ t.\ (\varphi\ t)\ \$\ i)$ *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*)) ⇒ ('*a* ^'*n*) **and** φ::*real* ⇒ ('*a* ^'*n*)
assumes ∀ *i*::'*n*. $((\lambda\ t.\ (\varphi\ t)\ \$\ i)$ *solves-ode* $(\lambda\ t\ s.\ (f\ (\varphi\ t))\ \$\ i))\ \{0..t\}$ *UNIV*
shows $(\varphi$ *solves-ode* $(\lambda\ t.\ f))\ \{0..t\}$ *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*]
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*:$((\ *\ )\ a$ *has-vector-derivative* $a)$ (*at x within T*)
by (*auto intro*: *derivative-eq-intros*)

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

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

lemma *quadratic-monomial-vderiv*[*poly-derivatives*]:$((\lambda t.\ a * t^2\ /\ 2)$ *has-vderiv-on*

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

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

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

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

2.5 Picard-Lindeloef

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*]

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

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

and *global-lipschitz-def* [*ubc-definitions*]
and *interval-def* [*ubc-definitions*]
and *nonempty-set-def* [*ubc-definitions*]

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

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

locale *picard-ivp* =
fixes f::*real* $\Rightarrow$ ($'a$::*banach*) $\Rightarrow$ $'a$ **and** T::*real set* **and** S::$'a$ *set* **and** L $t0$::*real*
assumes *init-time*:$t0 \in T$
and *cont-vec-field*: *continuous-on* $(T \times X)$ $(\lambda(t, x).\ f\ t\ x)$
and *lipschitz-vec-field*: $\bigwedge t.\ t \in T \Longrightarrow L$*−lipschitz-on* X $(\lambda x.\ f\ t\ x)$
and *nonempty-time*: $T \neq \{\}$
and *interval-time*: *is-interval* T
and *compact-time*: *compact* T
and *lipschitz-bound*: $\bigwedge s\ t.\ s \in T \Longrightarrow t \in T \Longrightarrow abs\ (s - t) * L < 1$
and *closed-domain*: *closed* S
and *solution-in-domain*:$\bigwedge x\ s\ t.\ t \in T \Longrightarrow x\ t0 = s \Longrightarrow x \in \{t0{-}{-}t\} \to S \Longrightarrow$
continuous-on $\{t0{-}{-}t\}$ $x \Longrightarrow x\ t0 +$ *ivl-integral* $t0\ t$ $(\lambda t.\ f\ t\ (x\ t)) \in S$
begin

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

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

sublocale *closed-domain S*
using *closed-domain* **unfolding** *closed-domain-def* **by** *simp*

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

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-vec-field lipschitz-bound cont-vec-field)

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

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

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

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 ((λ *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* (λ *t*. *f*) *T S L 0* **for** *f*::(*'a*::*banach*) ⇒ *'a* **and** *T S L* +
  fixes φ :: *real* ⇒ *'a* ⇒ *'a*
  assumes *ivp*:∀ *s* ∈ *S*. ((λ*t*. φ *t s*) *solves-ode* (λ *t*. *f*))*T S* ∧ φ *0 s* = *s*
begin

lemma *is-fixed-point*:
  assumes *s* ∈ *S* **and** *t* ∈ *T*
  shows φ *t s* = *phi t s*
  apply(*rule fixed-point-usolves*)
  using *ivp assms init-time* **by** *simp-all*

theorem *solves*:
  assumes *s* ∈ *S*
  shows ((λ *t*. φ *t s*) *solves-ode* (λ *t*. *f*))*T S*
  using *assms init-time fixed-point-solves*(*1*) **and** *is-fixed-point* **by** *auto*

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

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

lemma *usolves*:
  assumes (*x solves-ode* (λ*t*. *f*))*T S* **and** *x 0* = *s* **and** *t* ∈ *T*
  shows *x t* = φ *t s*
proof−
  from *assms* **and** *fixed-point-usolves*
  have *x t* = *phi t s* **by** *blast*
  also have ... = φ *t s* **using** *assms is-fixed-point*
    init-time solves-ode-domainD **by** *force*
  finally show *?thesis* **.**
qed

lemma *usolves-on-compact-subset*:
  assumes *T'* ⊆ *T* **and** *compact-interval T'* **and** *0* ∈ *T'*
  shows *t* ∈ *T'* ⟹ (*x solves-ode* (λ*t*. *f*)) *T' S* ⟹ φ *t* (*x 0*) = *x t*
proof−
  fix *t* **and** *x* **assume** *t* ∈ *T'* **and** *x-solves*:(*x solves-ode* (λ*t*. *f*))*T' S*
  from *this* **and** ‹*0* ∈ *T'*› **have** *x 0* ∈ *S* **unfolding** *solves-ode-def* **by** *blast*
  then have ((λ τ. φ τ (*x 0*)) *solves-ode* (λ τ. *f*))*T S* **using** *solves* **by** *blast*

hence *flow-solves*:$((\lambda\ \tau.\ \varphi\ \tau\ (x\ 0))$ *solves-ode* $(\lambda\ \tau.\ f))T'\ S$
  using ‹$T' \subseteq T$› *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** ‹$x\ 0 \in S$› **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*
  ‹$0 \in T'$› ‹$x\ 0 \in S$› ‹$T' \subseteq T$› **and** ‹*compact-interval* T'› **by** *blast*
moreover have $\varphi\ 0\ (x\ 0) = x\ 0$
  using *on-init-time* **and** ‹$x\ 0 \in S$› **by** *blast*
ultimately show $\varphi\ t\ (x\ 0) = x\ t$
  using *unique-on-bounded-closed.unique-solution flow-solves x-solves* **and** ‹$t \in T'$› **by** *blast*
qed

end

lemma *flow-on-compact-subset*:
assumes *flow-on-big*:*local-flow* $f\ T'\ S\ L\ \varphi$ **and** $T \subseteq T'$ **and** *compact-interval* T **and** $0 \in T$
shows *local-flow* $f\ T\ S\ L\ \varphi$
unfolding *local-flow-def local-flow-axioms-def* **proof**(*safe*)
fix s **show** $s \in S \Longrightarrow ((\lambda t.\ \varphi\ t\ s)$ *solves-ode* $(\lambda t.\ f))\ T\ S\ s \in S \Longrightarrow \varphi\ 0\ s = s$
  using *assms solves-ode-on-subset* **unfolding** *local-flow-def local-flow-axioms-def* **by** *fastforce+*
next
show *picard-ivp* $(\lambda t.\ f)\ T\ S\ L\ 0$
  using *assms* **unfolding** *local-flow-def local-flow-axioms-def*
  picard-ivp-def ubc-definitions **apply** *safe*
  apply(*meson Sigma-mono continuous-on-subset subsetI*)
  apply(*simp-all add*: *subset-eq*)
  by *fastforce*
qed

The last locale shows that the function introduced in its predecesor 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* $L\ \varphi$ **for** $f\ L\ \varphi$
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 *is-group-action*:

```
  shows φ 0 s = s
    and φ (t1 + t2) s = φ t1 (φ t2 s)
proof−
  show φ 0 s = s using on-init-time by simp
  have g1:φ (0 + t2) s = φ t2 s by simp
  have g2:((λ τ. φ (τ + t2) s) solves-ode (λ t. f))UNIV UNIV
    using add-flow-solves by simp
  have h0:φ t2 s ∈ UNIV
    using is-banach-endo by simp
  hence h1:φ 0 (φ t2 s) = φ t2 s
    using on-init-time by simp
  have h2:((λ τ. φ τ (φ t2 s)) solves-ode (λ t. f))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 ⋀ t. φ (t + t2) s = φ t (φ t2 s)
    using unique-on-bounded-closed.unique-solution is-ubc by blast
  thus φ (t1 + t2) s = φ t1 (φ t2 s) by simp
qed

end

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

2.5.1 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 ≤ t ⟹ picard-ivp (λt s. c) {0..t} UNIV (1 / (t
+ 1)) 0
  unfolding picard-ivp-def by(simp add: nonempty-set-def lipschitz-on-def, clar-
simp, simp)

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

lemma line-is-local-flow:
0 ≤ t ⟹ local-flow (λ s. (c::'a::banach)) {0..t} UNIV (1/(t + 1)) (λ t x. x + t
*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*
imports
../hs-prelims
../../afpModified/VC-KAD

begin

3 Hybrid System Verification

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

definition *f2r* :: $('a \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set\ (\mathcal{R})$ **where**
$\mathcal{R}\ f = \{(x,y).\ y \in f\ x\}$

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

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

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

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

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

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

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

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

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

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

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 Verification by providing solutions

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

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*} ≡ $\mathcal{R}$ (λ *s*. *g-orbital f T S t0 s G*)

context *picard-ivp*
begin

lemma *orbital-collapses*:
  assumes *ivp*:$\forall s \in S.\ ((\lambda t.\ \varphi\ t\ s)$ *solves-ode* $f) T\ S \wedge \varphi\ t0\ s = s$ **and** $s \in S$
  shows *orbital* $f\ T\ S\ t0\ s = \{\varphi\ t\ s |\ t.\ t \in T\}$
  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*:$\forall s \in S.\ ((\lambda t.\ \varphi\ t\ s)$ *solves-ode* $f) T\ S \wedge \varphi\ t0\ s = s$ **and** $s \in S$
  shows *g-orbital* $f\ T\ S\ t0\ s\ G = \{\varphi\ t\ s |\ t.\ t \in T \wedge (\forall\ r \in \{t0{-}{-}t\}.\ G\ (\varphi\ r\ s))\}$
  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*:$\forall s \in S.\ ((\lambda t.\ \varphi\ t\ s)$ *solves-ode* $f) T\ S \wedge \varphi\ t0\ s = s$
  shows *wp* $(\mathcal{R}\ (\lambda\ s.$ *orbital* $f\ T\ S\ t0\ s))\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall\ t \in T.\ s \in S \longrightarrow Q\ (\varphi\ t$
$s) \rceil$
  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* $\{[x' {=} f] T\ S\ @\ t0\ \&\ G\}\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall\ t \in T.\ s \in S \longrightarrow (\forall\ r \in$
$\{t0{-}{-}t\}. G\ (\varphi\ r\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil$
  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*)
  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 $\lceil P \rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ t0\ \&\ G\})\ \lceil Q \rceil$
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*: $0 \le t \Longrightarrow wp\ \{[x'=\lambda t\ s.\ c]\{0..t\}\ UNIV\ @\ 0\ \&\ G\}\ \lceil Q \rceil = \lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall r{\in}\{0{-}{-}\tau\}.\ G\ (x + r *_R c)) \longrightarrow Q\ (x + \tau *_R c) \rceil$
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*

3.3 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.

3.3.1 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*)

3.3.2 Differential Cut

lemma *wp-g-orbit-IdD*:
assumes $wp\ (\{[x'=f]T\ S\ @\ t0\ \&\ G\})\ \lceil C \rceil = Id$ **and** $\forall\ r{\in}\{t0{-}{-}t\}.\ (a,\ x\ r) \in \{[x'=f]T\ S\ @\ t0\ \&\ G\}$
shows $\forall r{\in}\{t0{-}{-}t\}.\ C\ (x\ r)$
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 $r \notin \{t0--t\} \vee C\ (x\ r)$ **using** *assms* **by** *blast*}
then show *?thesis* **by** *blast*
qed

theorem *DC*:
assumes $t0 \in T$ **and** *interval* T
and $wp\ (\{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\})\ \lceil C\rceil = Id$
shows $wp\ (\{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\})\ \lceil Q\rceil = wp\ \{[x\acute{}=f]T\ S\ @\ t0\ \&\ \lambda s.\ G\ s \wedge C\ s\}\ \lceil Q\rceil$
proof(*rule-tac* $f{=}\lambda\ x.\ wp\ x\ \lceil Q\rceil$ **in** *HOL.arg-cong*, *safe*)
fix $a\ b$ **assume** $(a,\ b) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\}$
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** $b = x\ t$
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*) ‹$t \in T$› *interval.closed-segment-subset-domain* **by** *blast*
ultimately have $\forall r{\in}\{t0--t\}.\ (a,\ x\ r) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\}$
using *x-solves* ‹$x\ t0 = a$› ‹$a \in S$› **unfolding** *f2r-def* **by** *blast*
from *this* **have** $\forall r{\in}\{t0--t\}.\ C\ (x\ r)$ **using** *wp-g-orbit-IdD assms*(*3*) **by** *blast*
thus $(a,\ b) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ \lambda s.\ G\ s \wedge C\ s\}$ **unfolding** *f2r-def*
using *guard-x* ‹$a \in S$› ‹$b = x\ t$› ‹$t \in T$› ‹$x\ t0 = a$› *x-solves* ‹$\forall r{\in}\{t0--t\}.\ r \in T$› **by** *fastforce*
next
fix $a\ b$ **assume** $(a,\ b) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ \lambda s.\ G\ s \wedge C\ s\}$
then show $(a,\ b) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\}$
unfolding *f2r-def* **by** *blast*
qed

theorem *dCut*:
assumes $t0 \in T$ **and** *interval* T
and *wp-C*:$\lceil P\rceil \subseteq wp\ (\{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\})\ \lceil C\rceil$
and *wp-Q*:$\lceil P\rceil \subseteq wp\ (\{[x\acute{}=f]T\ S\ @\ t0\ \&\ (\lambda\ s.\ G\ s \wedge C\ s)\})\ \lceil Q\rceil$
shows $\lceil P\rceil \subseteq wp\ (\{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\})\ \lceil Q\rceil$
proof(*subst wp-rel*, *simp add*: *p2r-def*, *clarsimp*)
fix $a\ y$ **assume** $P\ a$ **and** $(a,\ y) \in \{[x\acute{}=f]T\ S\ @\ t0\ \&\ G\}$
then obtain $x\ t$ **where** $t \in T$ **and** *x-solves*:(x *solves-ode* f) $T\ S$ **and** $x\ t = y$
and $x\ t0 = a$ **and** *guard-x*:$(\forall\ r \in \{t0--t\}.\ G\ (x\ r))$ **and** $a \in S$ **by**(*auto simp*: *f2r-def*)
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*) ‹$t \in T$› *interval.closed-segment-subset-domain* **by** *blast*

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

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

3.3.3 Differential Invariant

lemma *DI-sufficiency*:
assumes *picard*:*picard-ivp f T S L t0*
shows $wp\ \{[x'=f]T\ S\ @\ t0\ \&\ G\}\ \lceil Q\rceil \subseteq wp\ \lceil G\rceil\ \lceil\lambda s.\ s \in S \longrightarrow Q\ s\rceil$
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** $(x\ \textit{solves-ode}\ f)T\ S \wedge x\ t0 = s$
using *picard-ivp.fixed-point-solves* **by** *blast*
then also have $\forall\ r \in \{t0--t0\}.\ G\ (x\ r)$ **using** ‹$G\ s$› **by** *simp*
ultimately have $(s,s) \in \{[x'=f]T\ S\ @\ t0\ \&\ G\}$
using *picard picard-ivp.init-time* ‹$s \in S$› *f2r-def* **by** *fastforce*
thus $Q\ s$ **using** *wlpQ* **by** *blast*
qed

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

lemma *dInvariant*:
fixes $\vartheta::'a::banach \Rightarrow real$
assumes $\lceil G\rceil \subseteq \lceil I'\rceil$ **and** I' *is-pderivative-of* I *with-respect-to* $f\ T\ S$
shows $\lceil I\rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ (\textit{Inf}\ T)\ \&\ G\})\ \lceil I\rceil$
using *assms* **unfolding** *pderivative-def* **apply**(*subst wp-rel*)
proof(*simp add*: *p2r-def*, *clarsimp*)
assume *prime*:$\forall x.\ (x\ \textit{solves-ode}\ f)\ T\ S \longrightarrow I\ (x\ (\textit{Inf}\ T)) \longrightarrow (\forall t\in T.\ (\forall r\in\{\textit{Inf}$

$T{-}{-}t\}$. I' $(x\ r)) \longrightarrow I\ (x\ t))$
fix *s y* **assume** $(s,y) \in \{[x\acute{}=f]T\ S\ @\ (Inf\ T)\ \&\ G\}$ **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*:$\forall r{\in}\{(Inf\ T){-}{-}t\}.\ G\ (x\ r)$
by(*simp add*: *f2r-def*, *clarify*, *auto*)
hence $(Inf\ T) \leq t$ **by** (*simp add*: ‹*bdd-below T*› *cInf-lower*)
from *GHyp* **and** $\langle\lceil G\rceil \subseteq \lceil I'\rceil\rangle$ **have** *geq0*:$\forall r{\in}\{(Inf\ T){-}{-}t\}.\ I'\ (x\ r)$
by (*auto simp*: *p2r-def*)
thus $I\ y$ **using** *xtHyp x-ivp sHyp* **and** *prime* **by** *blast*
qed

lemma *invariant-eq-0*:
fixes $\vartheta{::}'a{::}banach \Rightarrow real$
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 $\lceil G\rceil \subseteq \lceil\lambda s.\ \nu\ s = 0\rceil$ **and** *bdd-below* T
shows $\lceil\lambda s.\ \vartheta\ s = 0\rceil \subseteq wp\ (\{[x\acute{}=f]T\ S\ @\ (Inf\ T)\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s = 0\rceil$
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* $f)\ T\ S\ \vartheta\ (x\ (Inf\ T)) = 0$
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*: ‹$(Inf\ T) \leq t$›)
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 $\lceil\lambda s.\ \vartheta\ s = 0\rceil \subseteq wp\ (\{[x\acute{}=f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s = 0\rceil$
apply(*subgoal-tac* $\lceil\lambda s.\ \vartheta\ s = 0\rceil \subseteq wp\ (\{[x\acute{}=f]\{t0..t\}\ S\ @\ (Inf\ \{t0..t\})\ \&\ G\})$ $\lceil\lambda s.\ \vartheta\ s = 0\rceil$)
apply(*subgoal-tac* $Inf\ \{t0..t\} = t0$, *simp*)
using ‹$t0 \leq t$› **apply** *simp*
apply(*rule invariant-eq-0*[*of* - $\{t0..t\}$ - - ν])
using *assms* **by**(*auto simp*: *closed-segment-eq-real-ivl*)

```
theorem dInvariant-eq-0:
  fixes ϑ::'a::banach ⇒ real and ν::'a ⇒ real
  assumes ∀x. (x solves-ode f) {t0..t} S ⟶
  (∀τ∈{t0..t}. ∀r∈{t0..τ}. ((λτ. ϑ (x τ)) has-derivative (λτ. τ *R ν (x r))) (at r
within {t0..τ}))
    and impls:⌈P⌉ ⊆ ⌈λs. ϑ s = 0⌉ ⌈λs. ϑ s = 0⌉ ⊆ ⌈Q⌉ ⌈G⌉ ⊆ ⌈λs. ν s = 0⌉
and t0 ≤ t
  shows ⌈P⌉ ⊆ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈Q⌉
  apply(rule-tac C=λs. ϑ s = 0 in dCut-interval, simp add: ‹t0 ≤ t›)
   apply(subgoal-tac ⌈λs. ϑ s = 0⌉ ⊆ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈λs. ϑ s
= 0⌉)
  using impls apply blast
   apply(rule-tac ν=ν 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 ϑ::'a::banach ⇒ real
  assumes nuHyp:∀ x. (x solves-ode f) T S ⟶ (∀ t ∈ T. ∀ r ∈ {(Inf T)−−t}.
  ((λτ. ϑ (x τ)) has-derivative (λτ. τ *R (ν (x r)))) (at r within {(Inf T)−−t}))
    and ⌈G⌉ ⊆ ⌈λs. (ν s) ≥ 0⌉ and bdd-below T
  shows ⌈λs. ϑ s ≥ 0⌉ ⊆ wp ({[x´=f]T S @ (Inf T) & G}) ⌈λs. ϑ s ≥ 0⌉
  apply(rule dInvariant [of - λ s. ν s ≥ 0])
  unfolding pderivative-def using assms apply(simp, simp)
proof(clarify)
  fix x and t
  assume x-ivp:ϑ (x (Inf T)) ≥ 0 (x solves-ode f) T S
    and tHyp:t ∈ T and ge0:∀r∈{Inf T−−t}. ν (x r) ≥ 0
  hence (Inf T) ≤ t by (simp add: ‹bdd-below T› cInf-lower)
  have ∀ r ∈ {(Inf T)−−t}. ((λτ. ϑ (x τ)) has-derivative (λτ. τ *R (ν (x r))))
    (at r within {(Inf T)−−t}) using nuHyp x-ivp(2) and tHyp by auto
  then have ∃r∈{(Inf T)−−t}. ϑ (x t)− ϑ (x (Inf T)) = (λτ. τ *R (ν (x r))) (t
− (Inf T))
    by(rule-tac closed-segment-mvt, auto simp: ‹(Inf T) ≤ t›)
  from this obtain r where
    r ∈ {(Inf T)−−t} ∧ ϑ (x t)= (t − Inf T) *R ν (x r) + ϑ (x (Inf T))  by force

  thus 0 ≤ ϑ (x t) by (simp add: ‹Inf T ≤ t› ge0 x-ivp(1))
qed

corollary invariant-geq-0-interval:
  fixes ϑ::'a::banach ⇒ real
  assumes ∀ x. (x solves-ode f){t0..t} S ⟶ (∀ τ ∈ {t0..t}. ∀ r ∈ {t0..τ}.
  ((λτ. ϑ (x τ)) has-derivative (λτ. τ *R (ν (x r)))) (at r within {t0..τ}))
    and ⌈G⌉ ⊆ ⌈λs. ν s ≥ 0⌉ and t0 ≤ t
  shows ⌈λs. ϑ s ≥ 0⌉ ⊆ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈λs. ϑ s ≥ 0⌉
  apply(subgoal-tac ⌈λs. ϑ s ≥ 0⌉ ⊆ wp ({[x´=f]{t0..t} S @ (Inf {t0..t}) & G})
⌈λs. ϑ s ≥ 0⌉)
```

apply(*subgoal-tac Inf* $\{t0..t\} = t0$, *simp*)
using ‹$t0 \leq t$› **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 ϑ::$'a$::*banach* $\Rightarrow$ *real* **and** ν::$'a \Rightarrow$ *real*
assumes $\forall x$. (x *solves-ode* f) $\{t0..t\}$ $S \longrightarrow$
($\forall \tau\in\{t0..t\}$. $\forall r\in\{t0..\tau\}$. (($\lambda\tau$. ϑ (x τ)) *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*: ‹$t0 \leq t$›)
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 ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes *nuHyp*:$\forall\ x$. (x *solves-ode* f) T $S \longrightarrow$ ($\forall\ t \in T$. $\forall\ r \in \{(Inf\ T)--t\}$.
(($\lambda\tau$. ϑ (x τ)) *has-derivative* ($\lambda\tau$. $\tau *_R$ (ν (x r)))) (*at* r *within* $\{(Inf\ T)--t\}$))
and $\lceil G\rceil \subseteq \lceil\lambda s.\ (\nu\ s) \leq 0\rceil$ **and** *bdd-below* T
shows $\lceil\lambda s.\ \vartheta\ s \leq 0\rceil \subseteq$ *wp* ($\{[x'=f]T$ S @ (*Inf* T) & $G\}$) $\lceil\lambda s.\ \vartheta\ s \leq 0\rceil$
apply(*rule dInvariant* [*of* - λ s. ν $s \leq 0$])
unfolding *pderivative-def* **using** *assms* **apply**(*simp*, *simp*)
proof(*clarify*)
fix x **and** t
assume *x-ivp*:ϑ (x (*Inf* T)) ≤ 0 (x *solves-ode* f) T S
and *tHyp*:$t \in T$ **and** *ge0*:$\forall r\in\{Inf\ T--t\}$. ν (x r) ≤ 0
hence (*Inf* T) $\leq t$ **by** (*simp add*: ‹*bdd-below* T› *cInf-lower*)
have $\forall\ r \in \{(Inf\ T)--t\}$. (($\lambda\tau$. ϑ (x τ)) *has-derivative* ($\lambda\tau$. $\tau *_R$ (ν (x r))))
(*at* r *within* $\{(Inf\ T)--t\}$) **using** *nuHyp* *x-ivp*(*2*) **and** *tHyp* **by** *auto*
then have $\exists r\in\{(Inf\ T)--t\}$. ϑ (x t)$-$ ϑ (x (*Inf* T)) $=$ ($\lambda\tau$. $\tau *_R$ (ν (x r))) (t $-$ (*Inf* T))
by(*rule-tac closed-segment-mvt*, *auto simp*: ‹(*Inf* T) $\leq t$›)
from *this* **obtain** r **where**
$r \in \{(Inf\ T)--t\} \wedge \vartheta$ (x t)$=$ (t $-$ *Inf* T) $*_R$ ν (x r) $+$ ϑ (x (*Inf* T)) **by** *force*
thus ϑ (x t) ≤ 0 **using** ‹(*Inf* T) $\leq t$› *ge0* *x-ivp*(*1*)
by (*metis add-decreasing2 ge-iff-diff-ge-0 split-scaleR-neg-le*)
qed

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

theorem *dInvariant-leq-0*:
fixes ϑ::'a::banach $\Rightarrow$ *real* **and** ν::'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*:$\lceil P\rceil \subseteq \lceil\lambda s.\ \vartheta\ s \leq 0\rceil$ $\lceil\lambda s.\ \vartheta\ s \leq 0\rceil \subseteq \lceil Q\rceil$ $\lceil G\rceil \subseteq \lceil\lambda s.\ \nu\ s \leq 0\rceil$
and $t0 \leq t$
shows $\lceil P\rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil Q\rceil$
apply(*rule-tac* $C=\lambda s.\ \vartheta\ s \leq 0$ **in** *dCut-interval*, *simp add*: $\langle t0 \leq t\rangle$)
apply(*subgoal-tac* $\lceil\lambda s.\ \vartheta\ s \leq 0\rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s$
$\leq 0\rceil)$
using *impls* **apply** *blast*
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** *simp*

lemma *invariant-above-0*:
fixes ϑ::'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 $\lceil\lambda s.\ \vartheta\ s > 0\rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ (Inf\ T)\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s > 0\rceil$
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-ivp*:$(x$ *solves-ode* $f)\ T\ S\ \vartheta\ (x\ (Inf\ T)) > 0$
and *tHyp*:$t \in T$ **and** *ge0*:$\forall r\in\{Inf\ T--t\}.\ \nu\ (x\ r) \geq 0$
hence $(Inf\ T) \leq t$ **by** (*simp add*: $\langle$*bdd-below T*$\rangle$ *cInf-lower*)
have $\forall\ r \in \{(Inf\ T)--t\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau))$ *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 $\exists r\in\{(Inf\ T)--t\}.\ \vartheta\ (x\ t) - \vartheta\ (x\ (Inf\ T)) = (\lambda\tau.\ \tau *_R (\nu\ (x\ r)))\ (t$
$- (Inf\ T))$
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 $0 < \vartheta\ (x\ t)$

by (*metis* ‹(*Inf T*) $\leq$ *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 ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall$ *x*. (*x solves-ode f*){*t0..t*} *S* $\longrightarrow$ ($\forall$ τ $\in$ {*t0..t*}. $\forall$ *r* $\in$ {*t0..*τ}.
(($\lambda\tau$. ϑ (*x* τ)) *has-derivative* ($\lambda\tau$. τ $*_R$ (ν (*x r*)))) (*at r within* {*t0..*τ}))
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*} - - ν])
using *assms* **by**(*auto simp*: *closed-segment-eq-real-ivl*)

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

lemma *invariant-below-0*:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes *nuHyp*:$\forall$ *x*. (*x solves-ode f*) *T S* $\longrightarrow$ ($\forall$ *t* $\in$ *T*. $\forall$ *r* $\in$ {(*Inf T*)−−*t*}.
(($\lambda\tau$. ϑ (*x* τ)) *has-derivative* ($\lambda\tau$. τ $*_R$ (ν (*x r*)))) (*at r within* {(*Inf T*)−−*t*}))
and $\lceil G\rceil \subseteq \lceil\lambda s.\ (\nu\ s) \leq 0\rceil$ **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* - λ *s*. ν *s* $\leq$ *0*])
unfolding *pderivative-def* **using** *assms* **apply**(*simp*, *simp*)
proof(*clarify*)
fix *x* **and** *t*
assume *x-ivp*:(*x solves-ode f*) *T S* ϑ (*x* (*Inf T*)) < *0*
and *tHyp*:$t \in T$ **and** *ge0*:$\forall r\in${*Inf T*−−*t*}. ν (*x r*) $\leq$ *0*
hence (*Inf T*) $\leq$ *t* **by** (*simp add*: ‹*bdd-below T*› *cInf-lower*)
have $\forall$ *r* $\in$ {(*Inf T*)−−*t*}. (($\lambda\tau$. ϑ (*x* τ)) *has-derivative* ($\lambda\tau$. τ $*_R$ (ν (*x r*))))

($at\ r\ within\ \{(Inf\ T)--t\}$) **using** *nuHyp x-ivp*(*1*) **and** *tHyp* **by** *auto*
then have $\exists\, r\in\{(Inf\ T)--t\}.\ \vartheta\ (x\ t)-\ \vartheta\ (x\ (Inf\ T)) = (\lambda\tau.\ \tau *_R (\nu\ (x\ r)))\ (t - (Inf\ T))$
by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$(Inf\ T) \le t$›)
thus $\vartheta\ (x\ t) < 0$ **using** ‹$(Inf\ T) \le t$› *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 ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (x\ solves\text{-}ode\ f)\{t0..t\}\ S \longrightarrow (\forall\ \tau \in \{t0..t\}.\ \forall\ r \in \{t0..\tau\}.$
$((\lambda\tau.\ \vartheta\ (x\ \tau))\ has\text{-}derivative\ (\lambda\tau.\ \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
and $\lceil G\rceil \subseteq \lceil\lambda s.\ \nu\ s \le 0\rceil$ **and** $t0 \le t$
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 ‹$t0 \le t$› **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 ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall x.\ (x\ solves\text{-}ode\ f)\ \{t0..t\}\ S \longrightarrow$
$(\forall\tau\in\{t0..t\}.\ \forall r\in\{t0..\tau\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau))\ has\text{-}derivative\ (\lambda\tau.\ \tau *_R \nu\ (x\ r)))\ (at\ r\ within\ \{t0..\tau\}))$
and *impls*: $\lceil P\rceil \subseteq \lceil\lambda s.\ \vartheta\ s < 0\rceil$ $\lceil\lambda s.\ \vartheta\ s < 0\rceil \subseteq \lceil Q\rceil$ $\lceil G\rceil \subseteq \lceil\lambda s.\ \nu\ s \le 0\rceil$ **and** $t0 \le t$
shows $\lceil P\rceil \subseteq wp\ (\{[x'=f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil Q\rceil$
using ‹$t0 \le t$› **apply**(*rule-tac* $C=\lambda s.\ \vartheta\ s < 0$ **in** *dCut-interval*, *simp add*: ‹$t0 \le t$›)
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-below-0-interval*)
using *assms*(*1*,*4*,*5*) **apply**(*simp*, *simp*, *simp*)
apply(*rule dWeakening*)
using *impls* **by** *simp*

lemma *invariant-meet*:
assumes $\lceil I1\rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ t0\ \&\ G\})\ \lceil I1\rceil$
and $\lceil I2\rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ t0\ \&\ G\})\ \lceil I2\rceil$
shows $\lceil\lambda s.\ I1\ s \wedge I2\ s\rceil \subseteq wp\ (\{[x'=f]T\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ I1\ s \wedge I2\ s\rceil$
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*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*I1*⌉ **and** ⌈*I2*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*I2*⌉
and *impls*:⌈*P*⌉ ⊆ ⌈λ*s*. *I1 s* ∧ *I2 s*⌉ ⌈λ*s*. *I1 s* ∧ *I2 s*⌉ ⊆ ⌈*Q*⌉ **and** *t0* ≤ *t*
shows ⌈*P*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*Q*⌉
apply(*rule-tac C*=λ*s*. *I1 s* ∧ *I2 s* **in** *dCut-interval*, *simp add*: ‹*t0* ≤ *t*›)
apply(*subgoal-tac* ⌈λ*s*. *I1 s* ∧ *I2 s*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s*. *I1 s* ∧ *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*⌉ ⊆ *wp* ({[*x´*=*f*]*T S* @ *t0* & *G*}) ⌈*I1*⌉
and ⌈*I2*⌉ ⊆ *wp* ({[*x´*=*f*]*T S* @ *t0* & *G*}) ⌈*I2*⌉
shows ⌈λ*s*. *I1 s* ∨ *I2 s*⌉ ⊆ *wp* ({[*x´*=*f*]*T S* @ *t0* & *G*}) ⌈λ*s*. *I1 s* ∨ *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 ⌈*I1*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*I1*⌉ **and** ⌈*I2*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*I2*⌉
and *impls*:⌈*P*⌉ ⊆ ⌈λ*s*. *I1 s* ∨ *I2 s*⌉ ⌈λ*s*. *I1 s* ∨ *I2 s*⌉ ⊆ ⌈*Q*⌉ **and** *t0* ≤ *t*
shows ⌈*P*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*Q*⌉
apply(*rule-tac C*=λ*s*. *I1 s* ∨ *I2 s* **in** *dCut-interval*, *simp add*: ‹*t0* ≤ *t*›)
apply(*subgoal-tac* ⌈λ*s*. *I1 s* ∨ *I2 s*⌉ ⊆ *wp* ({[*x´*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s*. *I1 s* ∨ *I2 s*⌉)
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

3.4 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.

3.4.1 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*:$0 \leq t \Longrightarrow t < 1 \Longrightarrow$
picard-ivp (λ *t s*. *free-fall-kinematics s*) {*0..t*} *UNIV 1 0*
unfolding *picard-ivp-def* **apply**(*simp add*: *lipschitz-on-def*, *safe*)
apply(*rule-tac t=X* **and** *f=snd* **in** *continuous-on-compose2*)
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 $0 \leq t$ **and** $t < 1$
shows *wp* {[*x´*=λ*t s*. *free-fall-kinematics s*]{*0..t*} *UNIV* @ *0* & *G*} $\lceil Q \rceil$ =
$\lceil\lambda$ *x*. $\forall$ $\tau \in$ {*0..t*}. ($\forall$ *r*$\in${*0−−*τ}. *G* (*free-fall-flow r x*)) $\longrightarrow$ *Q* (*free-fall-flow* τ *x*)$\rceil$
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 $0 \leq t$ **and** $t < 1$
shows
$\lceil\lambda$*s*. (*0*::*real*) $\leq$ *s* \$ (*Abs-three 0*) $\wedge$ *s* \$ (*Abs-three 0*) = *H* $\wedge$ *s* \$ (*Abs-three 1*) = *0* $\wedge$ *0* > *s* \$ (*Abs-three 2*)$\rceil$
$\subseteq$ *wp* ({[*x´*=λ*t s*. *free-fall-kinematics s*]{*0..t*} *UNIV* @ *0* & (λ *s*. *s* \$ (*Abs-three 0*) $\geq$ *0*)})
$\lceil\lambda$*s*. *0* $\leq$ *s* \$ (*Abs-three 0*) $\wedge$ *s* \$ (*Abs-three 0*) $\leq$ *H*$\rceil$
apply(*subst free-fall-flow-DS*)

by(*simp-all add*: *assms mult-nonneg-nonpos2*)

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

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

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

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-Lindeloef theorem:

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

lemma *picard-ivp-linear-system*:
fixes *A*::*real*^('*n*::*finite*)^'*n*
assumes *0* < ((*real CARD*('*n*))^2 · (*maxAbs A*)) (**is** *0* < *?L*)
assumes *0* ≤ *t* **and** *t* < *1*/*?L*
shows *picard-ivp* (λ *t s*. *A* ∗*v s*) {*0*..*t*} *UNIV ?L 0*
apply *unfold-locales* **apply**(*simp add*: ‹*0* ≤ *t*›)
subgoal by(*simp*, *metis continuous-on-compose2 continuous-on-cong continuous-on-id*

continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest)
subgoal using *matrix-lipschitz-constant 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*/((*real CARD*('*n*))^2 · *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* ≡ (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* ≡ (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-Lindeloef, 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* ≤ *t* ⟹ *t* < *1*/*9* ⟹
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 $wp\ \{[x' = \lambda t\ s.\ K *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ G\}\ \lceil Q \rceil =$
$\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall r \in \{0 -- \tau\}.\ G\ (\textit{flow-for-K}\ r\ x)) \longrightarrow Q\ (\textit{flow-for-K}\ \tau\ x) \rceil$
apply(*subst picard-ivp.wp-g-orbit*[*of* $\lambda t\ s.\ K *v\ s$ - - $((real\ CARD(3))^2 \cdot maxAbs\ K)$
- $(\lambda\ t\ x.\ \textit{flow-for-K}\ t\ x)$])
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 $\lceil \lambda s.\ (0::real) \leq s\ \$\ (0::3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2 \rceil$
$\subseteq wp\ (\{[x' = \lambda t\ s.\ K *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)\})\ \lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H \rceil$
apply(*subst flow-for-K-DS*)
using *assms* **by**(*simp-all add*: *mult-nonneg-nonpos2*)

3.4.3 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*]:$0 \leq x \Longrightarrow 0 > g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \Longrightarrow (x::real) \leq H$
proof–
assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*
hence $*$:$v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
from *this* **have** $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*
also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$
using *divide-nonneg-neg* **by** *fastforce*
ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma [*bb-real-arith*]:
assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
and *pos*:$g \cdot \tau^2\ /\ 2 + v \cdot \tau + (x::real) = 0$
shows $2 \cdot g \cdot H + (-\ (g \cdot \tau) - v) \cdot (-\ (g \cdot \tau) - v) = 0$

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 $\lceil \lambda s.\ (0{::}real) \leq s \ \$ \ (0{::}3) \wedge s \ \$ \ 0 = H \wedge s \ \$ \ 1 = 0 \wedge 0 > s \ \$ \ 2 \rceil \subseteq wp$
$((\{[x' {=} \lambda t\ s.\ K *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)\};$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1\ ::== (\lambda s.\ - s\ \$\ 1)])\ ELSE\ Id\ FI))^*)$
$\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H \rceil$
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*)

unfolding *rel-antidomain-kleene-algebra.cond-def rel-antidomain-kleene-algebra.ads-d-def*

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

3.4.4 Bouncing Ball with invariants

lemma *gravity-is-invariant*:(x *solves-ode* ($\lambda t.$ ($*v$) K)) $\{0..t\}$ *UNIV* $\Longrightarrow \tau \in \{0..t\} \Longrightarrow r \in \{0..\tau\} \Longrightarrow$
$((\lambda\tau.\ x\ \tau\ \$\ 2)$ *has-derivative* $(\lambda\tau.\ \tau *_R 0))$ (*at* r *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 *solves-ode* ($\lambda t.$ ($*v$) K)) $\{0..t\}$ *UNIV* $\Longrightarrow \tau \in \{0..t\} \Longrightarrow$
$r \in \{0..\tau\} \Longrightarrow ((\lambda\tau.\ 2 \cdot x\ \tau\ \$\ 2 \cdot x\ \tau\ \$\ 0 - 2 \cdot x\ \tau\ \$\ 2 \cdot H - x\ \tau\ \$\ 1 \cdot x\ \tau\ \$\ 1)$ *has-derivative*
$(\lambda\tau.\ \tau *_R 0))$ (*at* r *within* $\{0..\tau\}$)
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*=$\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$
and *g'1*=$\lambda t.\ 2 \cdot (t \cdot (x\ r\ \$\ 1 \cdot x\ r\ \$\ 2))$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac f'1*=$\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$ **and** *g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac f'1*=$\lambda t.\ 0$ **and** *g'1*=$(\lambda xa.\ xa \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*12*))
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*), *simp-all add*: *has-derivative-within-subset*)
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*7*))
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*), *simp-all add*: *has-derivative-within-subset*)
by(*rule-tac f'1*=$(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **and** *g'1*=$(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **in** *derivative-eq-intros*(*12*),

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

lemma *bouncing-ball-invariants*:
assumes $0 \leq t$ **and** $t < 1/9$
shows$\lceil\lambda s.\ (0{::}real) \leq s\ \$\ (0{::}3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil \subseteq wp$
$((\{[x\acute{}=\lambda t\ s.\ K *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)\};$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s. - s\ \$\ 1)])\ ELSE\ Id\ FI))^*)$
$\lceil\lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H\rceil$
apply(*rule-tac I*=$\lceil\lambda s.\ 0 \leq s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1)\rceil$ **in** *rel-ad-mka-starI*)
apply(*simp*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
apply(*subst p2r-r2p-wp-sym*[*of* $(IF\ (\lambda s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s. - s\ \$\ 1)])\ ELSE\ Id\ FI)$])
using *assms*(*1*) **apply**(*rule dCut-interval*[*of - - - - - -* $\lambda\ s.\ s\ \$\ 2 < 0$])
apply(*rule-tac* ϑ=$\lambda s.\ s\ \$\ 2$ **and** ν=$\lambda s.\ 0$ **in** *dInvariant-below-0*)

using *gravity-is-invariant* **apply** *force*
apply(*simp*, *simp*, *simp*, *simp add*: ‹$0 \leq t$›)
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*: ‹$0 \leq t$›)
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*: ‹$0 \leq t$›)
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*)

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

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

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

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

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

lemma *flow-for-Circ-solves-Circ*: $((\lambda\ \tau.\ flow\text{-}for\text{-}Circ\ \tau\ s)\ solves\text{-}ode\ (\lambda t\ s.\ Circ *v\ s))\ \{0..t\}\ UNIV$
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-distribs(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-distribs(4)*)
done
done

corollary *flow-for-Circ-DS*:
assumes *0 ≤ t* **and** *t < 1/4*
shows *wp {[x´=λ t s. Circ ∗v s]{0..t} UNIV @ 0 & G} ⌈Q⌉ =*
⌈λ x. ∀ τ ∈ {0..t}. (∀ r∈{0−−τ}. G (flow-for-Circ r x)) ⟶ Q (flow-for-Circ τ x)⌉
apply(*subst picard-ivp.wp-g-orbit[of λt s. Circ ∗v s - - ((real CARD(2))*^{2} *· max-Abs Circ) - (λ t x. 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 ≤ t* **and** *t < 1/4* **and** *(R::real) > 0*
shows⌈*λs. R*^{2} *= (s \$ (0::2))*^{2} *+ (s \$ 1)*^{2}⌉ *⊆ wp*
{[x´=λt s. Circ ∗v s]{0..t} UNIV @ 0 & (λ s. True)}
⌈*λs. R*^{2} *= (s \$ (0::2))*^{2} *+ (s \$ 1)*^{2}⌉
apply(*subst flow-for-Circ-DS*)
using *assms* **by** *simp-all*

end
theory *cat2funcset*
imports *../hs-prelims Transformer-Semantics.Kleisli-Quantale KAD.Modal-Kleene-Algebra*

begin

4 Hybrid System Verification

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

notation *Abs-nd-fun* ($-^{\bullet}$ [*101*] *100*) **and** *Rep-nd-fun* ($-_{\bullet}$ [*101*] *100*)
type-synonym $'a$ *pred* = $'a \Rightarrow$ *bool*

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

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

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

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

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

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

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

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

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

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

4.2 Weakest Liberal Preconditions

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

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

lemma *eq-p2ndf-iff* :$(\lceil P\rceil = \lceil Q\rceil) = (P = Q)$
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* :: $'a\ nd\text{-}fun \Rightarrow 'a \Rightarrow bool\ ((1\lfloor\text{-}\rfloor))$

where $\lfloor f \rfloor \equiv (\lambda x.\ x \in \mathit{Domain}\ (\mathcal{R}\ (f_\bullet)))$

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

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

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

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

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

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

lemma *wp-trafo*: $\lfloor \mathit{wp}\ F\ \lceil Q \rceil \rfloor = (\lambda s.\ \forall s'.\ s' \in (F_\bullet)\ s \longrightarrow Q\ s')$
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* :: $('a\,\hat{}\,'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\,\hat{}\,'b$ (-(*2*[- :== -]) [*70*, *65*] *61*)
where
$x[i :== a] \equiv (\chi\ j.\ (\mathit{if}\ j = i\ \mathit{then}\ a\ \mathit{else}\ (x\ \$\ j)))$

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

lemma *wp-assign*[*simp*]: *wp* $([x ::== \mathit{expr}])\ \lceil Q \rceil = \lceil \lambda s.\ Q\ (s[x :== \mathit{expr}\ s]) \rceil$
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* $\lceil P \rceil$ *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 $d\ p \leq |x^\star]\ q$
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*) (η *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*) = (λx. *if* ($f_\bullet$) *x* = {} *then* {} *else* η *x*
)$^\bullet$
unfolding *ads-d-def* **apply**(*rule nd-fun-ext*, *simp*)
apply *transfer* **by** *auto*

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

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

lemma *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 $d\ P \leq wp\ (F^\star)\ Q$
by(*simp add*: *fbox-starI*[*of - I*])
then show $P \leq wp$ (*qstar F*) *Q*
using ‹$d\ P = P$› **by** (*transfer*, *simp*)
qed

4.3 Verification by providing solutions

abbreviation *orbital f T S t0 x0* ≡
  $\{x\ t\ |t\ x.\ t \in T \wedge (x\ \textit{solves-ode}\ f)T\ S \wedge x\ t0 = x0 \wedge x0 \in S \wedge t0 \in T\}$
abbreviation *g-orbital f T S t0 x0 G* ≡
  $\{x\ t\ |t\ x.\ t \in T \wedge (x\ \textit{solves-ode}\ f)T\ S \wedge x\ t0 = x0 \wedge x0 \in S \wedge t0 \in T \wedge (\forall\ r \in \{t0{-}{-}t\}.\ G\ (x\ r))\}$

abbreviation
g-evolution ::(*real* ⇒ (′*a*::*banach*)⇒′*a*) ⇒ *real set* ⇒ ′*a set* ⇒ *real* ⇒ ′*a pred* ⇒ ′*a nd-fun*
((*1*{[*x*´=-]- - @ - & -})) **where** {[*x*´=*f*] *T S* @ *t0* & *G*} ≡ (λ *s*. *g-orbital f T S t0 s G*)$^\bullet$

context *picard-ivp*
begin

lemma *orbital-collapses*:
  assumes *ivp*:∀ *s* ∈ *S*. ((λ*t*. φ *t s*) *solves-ode f*)*T S* ∧ φ *t0 s* = *s* **and** *s* ∈ *S*
  shows *orbital f T S t0 s* = {φ *t s*| *t*. *t* ∈ *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*=λ*t*. φ *t s* **in** *exI*)
  using *assms init-time* **by** *auto*

lemma *g-orbital-collapses*:
  assumes *ivp*:∀ *s* ∈ *S*. ((λ*t*. φ *t s*) *solves-ode f*)*T S* ∧ φ *t0 s* = *s* **and** *s* ∈ *S*
  shows *g-orbital f T S t0 s G* = {φ *t s*| *t*. *t* ∈ *T* ∧ (∀ *r* ∈ {*t0*−−*t*}. *G* (φ *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*=λ*t*. φ *t s* **in** *exI*)
  using *assms init-time* **by** *auto*

lemma *wp-orbit*:
  assumes *ivp*:∀ *s* ∈ *S*. ((λ*t*. φ *t s*) *solves-ode f*)*T S* ∧ φ *t0 s* = *s*
  shows *wp* ((λ *s*. *orbital f T S t0 s*)$^\bullet$) ⌈*Q*⌉ = ⌈λ *s*. ∀ *t* ∈ *T*. *s* ∈ *S* ⟶ *Q* (φ *t s*)⌉
  apply(*subst wp-nd-fun*, *subst eq-p2ndf-iff*) **apply**(*rule ext*, *safe*)
   apply(*erule-tac x*=φ *t s* **in** *allE*, *erule impE*, *simp*)
    apply(*rule-tac x*=*t* **in** *exI*, *rule-tac x*=λ *t*. φ *t s* **in** *exI*)
  using *ivp init-time* **apply**(*simp*, *simp*)
  apply(*subgoal-tac* φ *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*:∀ *s* ∈ *S*. ((λ*t*. φ *t s*) *solves-ode f*)*T S* ∧ φ *t0 s* = *s*
  shows *wp* {[*x*´=*f*]*T S* @ *t0* & *G*} ⌈*Q*⌉ = ⌈λ *s*. ∀ *t* ∈ *T*. *s* ∈ *S* ⟶ (∀ *r* ∈ {*t0*−−*t*}.*G* (φ *r s*)) ⟶ *Q* (φ *t s*)⌉
  apply(*subst wp-nd-fun*, *subst eq-p2ndf-iff*) **apply**(*rule ext*, *safe*)

```
 apply(erule-tac x=φ t s in allE, erule impE, simp)
  apply(rule-tac x=t in exI, rule-tac x=λ t. φ t s in exI)
apply(simp add: ivp init-time, simp)
apply(subgoal-tac ∀r∈{t0−−t}. φ 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:∀ s ∈ S. ((λt. φ t s) solves-ode f) T S ∧
φ t0 s = s
    and ∀s. P s ⟶ (∀ t ∈ T. s ∈ S ⟶ (∀ r ∈ {t0..t}.G (φ r s)) ⟶ Q (φ t
s))
  shows ⌈P⌉ ≤ 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 ≤ t ⟹ wp {[x´=λt s. c]{0..t} UNIV @ 0 & G} ⌈Q⌉ =
    ⌈λ x. ∀ τ ∈ {0..t}. (∀r∈{0−−τ}. G (x + r *R c)) ⟶ Q (x + τ *R c)⌉
  apply(subst picard-ivp.wp-g-orbit[of λ t s. c - - 1/(t + 1) - (λ t x. x + t *R c)])
  using constant-is-picard-ivp apply blast
  using line-solves-constant by auto
```

4.4 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.

4.4.1 Differential Weakening

```
thm kcomp-def kcomp-prop le-fun-def

theorem DW:
  shows wp ({[x´=f]T S @ t0 & G}) ⌈Q⌉ = wp ({[x´=f]T S @ t0 & G}) ⌈λ s.
G s ⟶ Q s⌉
  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-orbital f T S t0 x G
```

have $*$:$\forall\, y \in\ ?Y.\ G\ y$ **by** *blast*
{assume $(\bigcup y{\in}?Y\ .\ \mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\}$
then have $\forall\, y{\in}?Y\ .\ (\mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\}$ **by** *blast*
hence $\forall\, y{\in}?Y\ .\ Q\ y$ **by** (*metis* (*mono-tags*, *lifting*) *bot-pres-del*)
then have $\forall\, y{\in}?Y\ .\ (\mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\}$ **by** *auto*
from *this* **have** $(\bigcup y{\in}?Y\ .\ \mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\}$ **by** *blast***}**
moreover
{assume $(\bigcup y{\in}?Y\ .\ \ \mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\}$
then have $\exists\, y{\in}?Y.\ (\mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\}$ **by** *blast*
hence $\exists\, y{\in}?Y.\ \neg\ Q\ y$ **by** (*metis* (*mono-tags*, *lifting*))
then have $\exists\, y{\in}?Y.\ (\mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\}$
by (*metis* (*mono-tags*, *lifting*) $*$ *bot-pres-del*)
from *this* **have** $(\bigcup y{\in}?Y.\ \mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\}$ **by** *blast***}**
ultimately show $((\bigcup y{\in}?Y.\ \mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\} \longrightarrow$
$(\bigcup y{\in}?Y.\ \mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) = \{\}) \wedge$
$((\bigcup y{\in}?Y.\ \mathit{if}\ \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\} \longrightarrow$
$(\bigcup y{\in}?Y.\ \mathit{if}\ G\ y \wedge \neg\ Q\ y\ \mathit{then}\ \eta\ y\ \mathit{else}\ \{\}) \neq \{\})$
by *blast*
qed

theorem *dWeakening*:
assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq \mathit{wp}\ (\{[x' {=} f]\, T\ S\ @\ t0\ \&\ G\})\ \lceil Q \rceil$
using *assms* **apply**(*subst wp-nd-fun*)
by(*auto simp*: *le-fun-def*)

4.4.2 Differential Cut

lemma *wp-g-orbit-etaD*:
assumes $\mathit{wp}\ (\{[x' {=} f]\, T\ S\ @\ t0\ \&\ G\})\ \lceil C \rceil = \eta^{\bullet}$ **and** $\forall\ r{\in}\{t0{-}{-}t\}.\ x\ r \in \mathit{g\text{-}orbital}\ f\ T\ S\ t0\ a\ G$
shows $\forall\, r{\in}\{t0{-}{-}t\}.\ C\ (x\ r)$
proof
fix r **assume** $r \in \{t0{-}{-}t\}$
then have $x\ r \in \mathit{g\text{-}orbital}\ f\ T\ S\ t0\ a\ G$
using *assms*(*2*) **by** *blast*
also have $\forall\, y.\ y \in (\mathit{g\text{-}orbital}\ f\ T\ S\ t0\ a\ G) \longrightarrow C\ y$
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 $\mathit{wp}\ (\{[x' {=} f]\, T\ S\ @\ t0\ \&\ G\})\ \lceil C \rceil = \eta^{\bullet}$
shows $\mathit{wp}\ (\{[x' {=} f]\, T\ S\ @\ t0\ \&\ G\})\ \lceil Q \rceil = \mathit{wp}\ (\{[x' {=} f]\, T\ S\ @\ t0\ \&\ \lambda s.\ G\ s \wedge C\ s\})\ \lceil Q \rceil$
proof(*rule-tac* $f{=}\lambda\ x.\ \mathit{wp}\ x\ \lceil Q \rceil$ **in** *HOL.arg-cong*, *rule nd-fun-ext*, *rule subset-antisym*, *simp-all*)
fix a

show *g-orbital f T S t0 a G* ⊆ *g-orbital f T S t0 a* (λ*s*. *G s* ∧ *C s*)
proof
fix *b* **assume** *b* ∈ *g-orbital f T S t0 a G*
then obtain *t*::*real* **and** *x* **where** *t* ∈ *T* **and** *x-solves*:(*x solves-ode f*) *T S*
and
x t0 = *a* **and** *guard-x*:(∀ *r*∈{*t0*−−*t*}. *G* (*x r*)) **and** *a* ∈ *S* **and** *b* = *x t*
using *assms*(*1*) **unfolding** *f2r-def* **by** *blast*
from *guard-x* **have** ∀ *r*∈{*t0*−−*t*}.∀ τ∈{*t0*−−*r*}. *G* (*x* τ)
using *assms*(*1*) **by** (*metis contra-subsetD ends-in-segment*(*1*) *subset-segment*(*1*))
also have ∀ *r*∈{*t0*−−*t*}. *r* ∈ *T*
using *assms*(*1*,*2*) ‹*t* ∈ *T*› *interval.closed-segment-subset-domain* **by** *blast*
ultimately have ∀ *r*∈{*t0*−−*t*}. *x r* ∈ *g-orbital f T S t0 a G*
using *x-solves* ‹*x t0* = *a*› ‹*a* ∈ *S*› **unfolding** *f2r-def* **by** *blast*
from *this* **have** ∀ *r*∈{*t0*−−*t*}. *C* (*x r*) **using** *wp-g-orbit-etaD assms*(*3*) **by**
blast
thus *b* ∈ *g-orbital f T S t0 a* (λ *s*. *G s* ∧ *C s*) **unfolding** *f2r-def*
using *guard-x* ‹*a* ∈ *S*› ‹*b* = *x t*› ‹*t* ∈ *T*› ‹*x t0* = *a*› *x-solves* ‹∀ *r*∈{*t0*−−*t*}. *r*
∈ *T*› **by** *fastforce*
qed
next show ⋀ *a*. *g-orbital f T S t0 a* (λ *s*. *G s* ∧ *C s*) ⊆ *g-orbital f T S t0 a G* **by**
auto
qed

theorem *dCut*:
assumes *t0* ∈ *T* **and** *interval T*
and *wp-C*:⌈*P*⌉ ≤ *wp* ({[*x*´=*f*]*T S* @ *t0* & *G*}) ⌈*C*⌉
and *wp-Q*:⌈*P*⌉ ≤ *wp* ({[*x*´=*f*]*T S* @ *t0* & (λ *s*. *G s* ∧ *C s*)}) ⌈*Q*⌉
shows ⌈*P*⌉ ≤ *wp* ({[*x*´=*f*]*T S* @ *t0* & *G*}) ⌈*Q*⌉
proof(*subst wp-nd-fun*, *clarsimp*)
fix *t*::*real* **and** *x*::*real* ⇒ ′*a* **assume** *P* (*x t0*) **and** *t* ∈ *T* **and** *x t0* ∈ *S*
and *x-solves*:(*x solves-ode f*)*T S* **and** *guard-x*:(∀ *r* ∈ {*t0*−−*t*}. *G* (*x r*))
from *guard-x* **have** ∀ *r*∈{*t0*−−*t*}.∀ τ∈{*t0*−−*r*}. *G* (*x* τ)
using ‹*t0* ∈ *T*› **by** (*metis contra-subsetD ends-in-segment*(*1*) *subset-segment*(*1*))

also have ∀ *r*∈{*t0*−−*t*}. *r* ∈ *T*
using ‹*t0* ∈ *T*› ‹*interval T*› ‹*t* ∈ *T*› *interval.closed-segment-subset-domain* **by**
blast
ultimately have ∀ *r*∈{*t0*−−*t*}. *x r* ∈ *g-orbital f T S t0* (*x t0*) *G*
using *x-solves* ‹*x t0* ∈ *S*› **by** *blast*
from *this* **have** ∀ *r*∈{*t0*−−*t*}. *C* (*x r*) **using** *wp-C* ‹*P* (*x t0*)› **by**(*subst* (*asm*)
wp-nd-fun, *simp*)
hence *x t* ∈ *g-orbital f T S t0* (*x t0*) (λ *s*. *G s* ∧ *C s*)
using *guard-x* ‹*t* ∈ *T*› *x-solves* ‹*x t0* ∈ *S*› ‹∀ *r*∈{*t0*−−*t*}. *r* ∈ *T*› **by** *fastforce*
from *this* ‹*P* (*x t0*)› **and** *wp-Q* **show** *Q* (*x t*)
by(*subst* (*asm*) *wp-nd-fun*, *simp*)
qed

corollary *dCut-interval*:
assumes *t0* ≤ *t* **and** ⌈*P*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*C*⌉

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

4.4.3 Differential Invariant

lemma *DI-sufficiency*:
assumes *picard-ivp f T S L t0*
shows *wp* $\{[x'=f]T\ S\ @\ t0\ \&\ G\}\ \lceil Q \rceil \leq$ *wp* $\lceil G \rceil\ \lceil \lambda s.\ s \in S \longrightarrow Q\ s \rceil$
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 ϑ::'a::*banach* $\Rightarrow$ *real*
assumes $\lceil G \rceil \leq \lceil I' \rceil$ **and** $t \geq 0$
and $\forall\ x.\ (x$ *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 $\lceil I \rceil \leq$ *wp* $(\{[x'=f]\{0..t\}\ S\ @\ 0\ \&\ G\})\ \lceil I \rceil$
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* $T \wedge (\forall\ x.\ (x$ *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 ϑ::'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 $\lceil G \rceil \leq \lceil \lambda s.\ \nu\ s = 0 \rceil$ **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 solves-ode f*) *T S* ϑ (*x* (*Inf T*)) = *0*
and *tHyp*:*t* ∈ *T* **and** *eq0*:∀ *r*∈{*Inf T*−−*t*}. ν (*x r*) = *0*
hence (*Inf T*) ≤ *t* **by** (*simp add*: ‹*bdd-below T*› *cInf-lower*)
have ∀ *r* ∈ {(*Inf T*)−−*t*}. ((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ ν (*x r*)))
(*at r within* {(*Inf T*)−−*t*}) **using** *nuHyp x-ivp*(*1*) **and** *tHyp* **by** *auto*
then have ∀ *r* ∈ {(*Inf T*)−−*t*}. ((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ *0*))
(*at r within* {(*Inf T*)−−*t*}) **using** *eq0* **by** *auto*
then have ∃ *r*∈{(*Inf T*)−−*t*}. ϑ (*x t*)− ϑ (*x* (*Inf T*)) = (λτ. τ $*_R$ *0*) (*t* − (*Inf T*))
by(*rule-tac closed-segment-mvt*, *auto simp*: ‹(*Inf T*) ≤ *t*›)
thus ϑ (*x t*) = *0*
using *x-ivp*(*2*) **by** (*metis right-minus-eq scale-zero-right*)
qed

corollary *invariant-eq-0-interval*:
fixes ϑ::′*a*::*banach* ⇒ *real*
assumes ∀ *x*. (*x solves-ode f*){*t0*..*t*} *S* ⟶ (∀ τ ∈ {*t0*..*t*}. ∀ *r* ∈ {*t0*..τ}.
((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ (ν (*x r*)))) (*at r within* {*t0*..τ}))
and ⌈*G*⌉ ≤ ⌈λ*s*. ν *s* = *0*⌉ **and** *t0* ≤ *t*
shows ⌈λ*s*. ϑ *s* = *0*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s*. ϑ *s* = *0*⌉
apply(*subgoal-tac* ⌈λ*s*. ϑ *s* = *0*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ (*Inf* {*t0*..*t*}) & *G*}) ⌈λ*s*. ϑ *s* = *0*⌉)
apply(*subgoal-tac Inf* {*t0*..*t*} = *t0*, *simp*)
using ‹*t0* ≤ *t*› **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 ϑ::′*a*::*banach* ⇒ *real* **and** ν::′*a* ⇒ *real*
assumes ∀ *x*. (*x solves-ode f*) {*t0*..*t*} *S* ⟶
(∀ τ∈{*t0*..*t*}. ∀ *r*∈{*t0*..τ}. ((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ ν (*x r*))) (*at r within* {*t0*..τ}))
and *impls*:⌈*P*⌉ ≤ ⌈λ*s*. ϑ *s* = *0*⌉ ⌈λ*s*. ϑ *s* = *0*⌉ ≤ ⌈*Q*⌉ ⌈*G*⌉ ≤ ⌈λ*s*. ν *s* = *0*⌉ **and** *t0* ≤ *t*
shows ⌈*P*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈*Q*⌉
apply(*rule-tac C*=λ*s*. ϑ *s* = *0* **in** *dCut-interval*, *simp add*: ‹*t0* ≤ *t*›)
apply(*subgoal-tac* ⌈λ*s*. ϑ *s* = *0*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s*. ϑ *s* = *0*⌉)
using *impls* **apply**(*subst* (*asm*) *wp-nd-fun*, *subst wp-nd-fun*) **apply** *auto*[*1*]
apply(*rule-tac* ν=ν **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 ϑ::′*a*::*banach* ⇒ *real*
assumes *nuHyp*:∀ *x*. (*x solves-ode f*) *T S* ⟶ (∀ *t* ∈ *T*. ∀ *r* ∈ {(*Inf T*)−−*t*}.

$((\lambda\tau.\ \vartheta\ (x\ \tau))$ *has-derivative* $(\lambda\tau.\ \tau *_R (\nu\ (x\ r))))$ $(at\ r\ within\ \{(Inf\ T){-}{-}t\})$)
and $\lceil G\rceil \leq \lceil\lambda s.\ (\nu\ s) \geq 0\rceil$ **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* - λ *s.* ν *s* $\geq$ *0*])
using *assms* **apply**(*simp*, *simp add*: *pderivative-def*)
proof(*clarify*)
fix *x* **and** *t*
assume *x-ivp*:$\vartheta\ (x\ (Inf\ T)) \geq 0\ (x\ solves\text{-}ode\ f)\ T\ S$
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*: ‹*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*(*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*: ‹$(Inf\ T) \leq t$›)
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 $0 \leq \vartheta\ (x\ t)$ **by** (*simp add*: ‹$Inf\ T \leq t$› *ge0 x-ivp*(*1*))
qed

corollary *invariant-geq-0-interval*:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (x\ solves\text{-}ode\ f)\{t0..t\}\ S \longrightarrow (\forall\ \tau \in \{t0..t\}.\ \forall\ r \in \{t0..\tau\}.$
$((\lambda\tau.\ \vartheta\ (x\ \tau))$ *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 $\lceil\lambda s.\ \vartheta\ s \geq 0\rceil \leq wp\ (\{[x'{=}f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s \geq 0\rceil$
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* $Inf\ \{t0..t\} = t0$, *simp*)
using ‹$t0 \leq t$› **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 ϑ::$'a$::*banach* $\Rightarrow$ *real* **and** ν::$'a \Rightarrow$ *real*
assumes $\forall x.\ (x\ solves\text{-}ode\ f)\ \{t0..t\}\ S \longrightarrow$
$(\forall \tau{\in}\{t0..t\}.\ \forall r{\in}\{t0..\tau\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau))$ *has-derivative* $(\lambda\tau.\ \tau *_R \nu\ (x\ r)))$ $(at\ r\ within\ \{t0..\tau\})$)
and *impls*:$\lceil P\rceil \leq \lceil\lambda s.\ \vartheta\ s \geq 0\rceil\ \lceil\lambda s.\ \vartheta\ s \geq 0\rceil \leq \lceil Q\rceil\ \lceil G\rceil \leq \lceil\lambda s.\ \nu\ s \geq 0\rceil$ **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*: ‹$t0 \leq t$›)
apply(*subgoal-tac* $\lceil\lambda s.\ \vartheta\ s \geq 0\rceil \leq wp\ (\{[x'{=}f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s \geq 0\rceil$)
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 ϑ::′a::banach ⇒ real
assumes *nuHyp*:∀ x. (x solves-ode f) T S ⟶ (∀ t ∈ T. ∀ r ∈ {(Inf T)−−t}.
((λτ. ϑ (x τ)) has-derivative (λτ. τ $*_R$ (ν (x r)))) (at r within {(Inf T)−−t}))
and ⌈G⌉ ≤ ⌈λs. (ν s) ≤ 0⌉ **and** bdd-below T
shows ⌈λs. ϑ s ≤ 0⌉ ≤ wp ({[x´=f]T S @ (Inf T) & G}) ⌈λs. ϑ s ≤ 0⌉
apply(rule dInvariant [of - λ s. ν s ≤ 0])
using assms **apply**(simp, simp add: pderivative-def)
proof(clarify)
fix x **and** t
assume *x-ivp*:ϑ (x (Inf T)) ≤ 0 (x solves-ode f) T S
and *tHyp*:t ∈ T **and** *ge0*:∀ r∈{Inf T−−t}. ν (x r) ≤ 0
hence (Inf T) ≤ t **by** (simp add: ‹bdd-below T› cInf-lower)
have ∀ r ∈ {(Inf T)−−t}. ((λτ. ϑ (x τ)) has-derivative (λτ. τ $*_R$ (ν (x r))))
(at r within {(Inf T)−−t}) **using** nuHyp x-ivp(2) **and** tHyp **by** auto
then have ∃ r∈{(Inf T)−−t}. ϑ (x t)− ϑ (x (Inf T)) = (λτ. τ $*_R$ (ν (x r))) (t − (Inf T))
by(rule-tac closed-segment-mvt, auto simp: ‹(Inf T) ≤ t›)
from this **obtain** r **where**
r ∈ {(Inf T)−−t} ∧ ϑ (x t)= (t − Inf T) $*_R$ ν (x r) + ϑ (x (Inf T)) **by** force
thus ϑ (x t) ≤ 0 **using** ‹(Inf T) ≤ t› ge0 x-ivp(1)
by (metis add-decreasing2 ge-iff-diff-ge-0 split-scaleR-neg-le)
qed

corollary *invariant-leq-0-interval*:
fixes ϑ::′a::banach ⇒ real
assumes ∀ x. (x solves-ode f){t0..t} S ⟶ (∀ τ ∈ {t0..t}. ∀ r ∈ {t0..τ}.
((λτ. ϑ (x τ)) has-derivative (λτ. τ $*_R$ (ν (x r)))) (at r within {t0..τ}))
and ⌈G⌉ ≤ ⌈λs. ν s ≤ 0⌉ **and** t0 ≤ t
shows ⌈λs. ϑ s ≤ 0⌉ ≤ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈λs. ϑ s ≤ 0⌉
apply(subgoal-tac ⌈λs. ϑ s ≤ 0⌉ ≤ wp ({[x´=f]{t0..t} S @ (Inf {t0..t}) & G}) ⌈λs. ϑ s ≤ 0⌉)
apply(subgoal-tac Inf {t0..t} = t0, simp)
using ‹t0 ≤ t› **apply**(simp add: closed-segment-eq-real-ivl)
apply(rule invariant-leq-0[of - {t0..t} - - ν])
using assms **by**(auto simp: closed-segment-eq-real-ivl)

theorem *dInvariant-leq-0*:
fixes ϑ::′a::banach ⇒ real **and** ν::′a ⇒ real
assumes ∀ x. (x solves-ode f) {t0..t} S ⟶
(∀ τ∈{t0..t}. ∀ r∈{t0..τ}. ((λτ. ϑ (x τ)) has-derivative (λτ. τ $*_R$ ν (x r))) (at r within {t0..τ}))
and *impls*:⌈P⌉ ≤ ⌈λs. ϑ s ≤ 0⌉ ⌈λs. ϑ s ≤ 0⌉ ≤ ⌈Q⌉ ⌈G⌉ ≤ ⌈λs. ν s ≤ 0⌉ **and** t0 ≤ t
shows ⌈P⌉ ≤ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈Q⌉
apply(rule-tac C=λs. ϑ s ≤ 0 **in** dCut-interval, simp add: ‹t0 ≤ t›)
apply(subgoal-tac ⌈λs. ϑ s ≤ 0⌉ ≤ wp ({[x´=f]{t0..t} S @ t0 & G}) ⌈λs. ϑ s ≤ 0⌉)

using *impls* **apply**(*subst* (*asm*) *wp-nd-fun*, *subst wp-nd-fun*) **apply** *auto*[*1*]
apply(*rule-tac* ν=ν **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 ϑ::'*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*}.
((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ (ν (*x r*)))) (*at r within* {(*Inf T*)−−*t*}))
and ⌈*G*⌉ ≤ ⌈λ*s*. (ν *s*) ≥ *0*⌉ **and** *bdd-below T*
shows ⌈λ*s*. ϑ *s* > *0*⌉ ≤ *wp* ({[*x*´=*f*]*T S* @ (*Inf T*) & *G*}) ⌈λ*s*. ϑ *s* > *0*⌉
apply(*rule dInvariant* [*of* - λ *s*. ν *s* ≥ *0*])
using *assms* **apply**(*simp*, *simp add*: *pderivative-def*)
proof(*clarify*)
fix *x* **and** *t*
assume *x-ivp*:(*x solves-ode f*) *T S* ϑ (*x* (*Inf T*)) > *0*
and *tHyp*:*t* ∈ *T* **and** *ge0*:∀*r*∈{*Inf T*−−*t*}. ν (*x r*) ≥ *0*
hence (*Inf T*) ≤ *t* **by** (*simp add*: ‹*bdd-below T*› *cInf-lower*)
have ∀ *r* ∈ {(*Inf T*)−−*t*}. ((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ (ν (*x r*))))
(*at r within* {(*Inf T*)−−*t*}) **using** *nuHyp x-ivp*(*1*) **and** *tHyp* **by** *auto*
then have ∃*r*∈{(*Inf T*)−−*t*}. ϑ (*x t*)− ϑ (*x* (*Inf T*)) = (λτ. τ $*_R$ (ν (*x r*))) (*t*
− (*Inf T*))
by(*rule-tac closed-segment-mvt*, *auto simp*: ‹(*Inf T*) ≤ *t*›)
from *this* **obtain** *r* **where**
r ∈ {(*Inf T*)−−*t*} ∧ ϑ (*x t*)= (*t* − *Inf T*) $*_R$ ν (*x r*) + ϑ (*x* (*Inf T*)) **by** *force*

thus *0* < ϑ (*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 ϑ::'*a*::*banach* $\Rightarrow$ *real*
assumes ∀ *x*. (*x solves-ode f*){*t0*..*t*} *S* ⟶ (∀ τ ∈ {*t0*..*t*}. ∀ *r* ∈ {*t0*..τ}.
((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ (ν (*x r*)))) (*at r within* {*t0*..τ}))
and ⌈*G*⌉ ≤ ⌈λ*s*. ν *s* ≥ *0*⌉ **and** *t0* ≤ *t*
shows ⌈λ*s*. ϑ *s* > *0*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s*. ϑ *s* > *0*⌉
apply(*subgoal-tac* ⌈λ*s*. ϑ *s* > *0*⌉ ≤ *wp* ({[*x*´=*f*]{*t0*..*t*} *S* @ (*Inf* {*t0*..*t*}) & *G*})
⌈λ*s*. ϑ *s* > *0*⌉)
apply(*subgoal-tac Inf* {*t0*..*t*} = *t0*, *simp*)
using ‹*t0* ≤ *t*› **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 ϑ::'*a*::*banach* $\Rightarrow$ *real* **and** ν::'*a* $\Rightarrow$ *real*
assumes ∀*x*. (*x solves-ode f*) {*t0*..*t*} *S* ⟶
(∀τ∈{*t0*..*t*}. ∀*r*∈{*t0*..τ}. ((λτ. ϑ (*x* τ)) *has-derivative* (λτ. τ $*_R$ ν (*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*: ‹$t0 \leq t$›)
 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-above-0-interval*)
 using *assms*(*1*,*4*,*5*) **apply**(*simp*, *simp*, *simp*)
 apply(*rule dWeakening*) **using** *impls* **by** *auto*

lemma *invariant-below-0*:
 fixes ϑ::*'a*::*banach* $\Rightarrow$ *real*
 assumes *nuHyp*:$\forall\ x.\ (x\ solves\text{-}ode\ f)\ T\ S \longrightarrow (\forall\ t \in T.\ \forall\ r \in \{(Inf\ T)\text{--}t\}.$
 $((\lambda\tau.\ \vartheta\ (x\ \tau))\ has\text{-}derivative\ (\lambda\tau.\ \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{(Inf\ T)\text{--}t\}))$
  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* - λ *s*. ν *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*:$\forall r \in \{Inf\ T\text{--}t\}.\ \nu\ (x\ r) \leq 0$
 hence $(Inf\ T) \leq t$ **by** (*simp add*: ‹*bdd-below T*› *cInf-lower*)
 have $\forall\ r \in \{(Inf\ T)\text{--}t\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau))\ has\text{-}derivative\ (\lambda\tau.\ \tau *_R (\nu\ (x\ r))))$
  $(at\ r\ within\ \{(Inf\ T)\text{--}t\})$ **using** *nuHyp x-ivp*(*1*) **and** *tHyp* **by** *auto*
 then have $\exists r \in \{(Inf\ T)\text{--}t\}.\ \vartheta\ (x\ t) - \vartheta\ (x\ (Inf\ T)) = (\lambda\tau.\ \tau *_R (\nu\ (x\ r)))\ (t$
$- (Inf\ T))$
  by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$(Inf\ T) \leq t$›)
 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 ϑ::*'a*::*banach* $\Rightarrow$ *real*
 assumes $\forall\ x.\ (x\ solves\text{-}ode\ f)\{t0..t\}\ S \longrightarrow (\forall\ \tau \in \{t0..t\}.\ \forall\ r \in \{t0..\tau\}.$
 $((\lambda\tau.\ \vartheta\ (x\ \tau))\ has\text{-}derivative\ (\lambda\tau.\ \tau *_R (\nu\ (x\ r))))\ (at\ r\ within\ \{t0..\tau\}))$
  and $\lceil G\rceil \leq \lceil\lambda s.\ \nu\ s \leq 0\rceil$ **and** $t0 \leq t$
 shows $\lceil\lambda s.\ \vartheta\ s < 0\rceil \leq wp\ (\{[x'=f]\{t0..t\}\ S\ @\ t0\ \&\ G\})\ \lceil\lambda s.\ \vartheta\ s < 0\rceil$
 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 ‹$t0 \leq t$› **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 ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall x$. (x *solves-ode* f) $\{t0..t\}$ $S \longrightarrow$
($\forall \tau \in \{t0..t\}$. $\forall r \in \{t0..\tau\}$. (($\lambda\tau$. ϑ (x τ)) *has-derivative* ($\lambda\tau$. $\tau *_R \nu$ (x r))) (*at* r *within* $\{t0..\tau\}$))
and *impls*:$\lceil P\rceil \le \lceil\lambda s.\ \vartheta\ s < 0\rceil$ $\lceil\lambda s.\ \vartheta\ s < 0\rceil \le \lceil Q\rceil$ $\lceil G\rceil \le \lceil\lambda s.\ \nu\ s \le 0\rceil$ **and** $t0 \le t$
shows $\lceil P\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil Q\rceil$
using ‹$t0 \le t$› **apply**(*rule-tac* C=λs. ϑ $s < 0$ **in** *dCut-interval*, *simp add*: ‹$t0 \le t$›)
apply(*subgoal-tac* $\lceil\lambda s.\ \vartheta\ s < 0\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ 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* ν=ν **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 $\lceil I1\rceil \le wp$ ($\{[x'=f]T$ S @ $t0$ & $G\}$) $\lceil I1\rceil$
and $\lceil I2\rceil \le wp$ ($\{[x'=f]T$ S @ $t0$ & $G\}$) $\lceil I2\rceil$
shows $\lceil\lambda s.\ I1\ s \wedge I2\ s\rceil \le wp$ ($\{[x'=f]T$ 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 $\lceil I1\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil I1\rceil$ **and** $\lceil I2\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil I2\rceil$
and *impls*:$\lceil P\rceil \le \lceil\lambda s.\ I1\ s \wedge I2\ s\rceil$ $\lceil\lambda s.\ I1\ s \wedge I2\ s\rceil \le \lceil Q\rceil$ **and** $t0 \le t$
shows $\lceil P\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil Q\rceil$
apply(*rule-tac* C=λs. $I1$ $s \wedge I2$ s **in** *dCut-interval*, *simp add*: ‹$t0 \le t$›)
apply(*subgoal-tac* $\lceil\lambda s.\ I1\ s \wedge I2\ s\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ 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 $\lceil I1\rceil \le wp$ ($\{[x'=f]T$ S @ $t0$ & $G\}$) $\lceil I1\rceil$
and $\lceil I2\rceil \le wp$ ($\{[x'=f]T$ S @ $t0$ & $G\}$) $\lceil I2\rceil$
shows $\lceil\lambda s.\ I1\ s \vee I2\ s\rceil \le wp$ ($\{[x'=f]T$ 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 $\lceil I1\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil I1\rceil$ **and** $\lceil I2\rceil \le wp$ ($\{[x'=f]\{t0..t\}$ S @ $t0$ & $G\}$) $\lceil I2\rceil$

and *impls*:⌈P⌉ ≤ ⌈λ*s. I1 s* ∨ *I2 s*⌉ ⌈λ*s. I1 s* ∨ *I2 s*⌉ ≤ ⌈Q⌉ **and** *t0* ≤ *t*
shows ⌈P⌉ ≤ *wp* ({[*x′*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈Q⌉
apply(*rule-tac C*=λ*s. I1 s* ∨ *I2 s* **in** *dCut-interval*, *simp add*: ‹*t0* ≤ *t*›)
apply(*subgoal-tac* ⌈λ*s. I1 s* ∨ *I2 s*⌉ ≤ *wp* ({[*x′*=*f*]{*t0*..*t*} *S* @ *t0* & *G*}) ⌈λ*s. I1 s* ∨ *I2 s*⌉)
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

4.5 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.

4.5.1 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*:$\forall\ x::three.\ x = Abs\text{-}three\ 0 \lor x = Abs\text{-}three\ 1 \lor x = Abs\text{-}three\ 2$
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*ˆ*three*) $\equiv$ (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* $\equiv$
(χ *i*. *if i*=(*Abs-three 0*) *then s* \$ (*Abs-three 2*) $\cdot$ *t* ˆ *2*/*2* + *s* \$ (*Abs-three 1*) $\cdot$ *t* + *s* \$ (*Abs-three 0*)
else if i=(*Abs-three 1*) *then s* \$ (*Abs-three 2*) $\cdot$ *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*:$0 \le t \Longrightarrow t < 1 \Longrightarrow$
picard-ivp (λ *t s*. *free-fall-kinematics s*) {*0*..*t*} *UNIV 1 0*
unfolding *picard-ivp-def* **apply**(*simp add*: *lipschitz-on-def*, *safe*)
apply(*rule-tac t*=*X* **and** *f*=*snd* **in** *continuous-on-compose2*)
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 $0 \leq t$ **and** $t < 1$
  shows $wp\ \{[x' {=} \lambda t\ s.\ \textit{free-fall-kinematics}\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ G\}\ \lceil Q \rceil =$
    $\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall r {\in} \{0{-}{-}\tau\}.\ G\ (\textit{free-fall-flow}\ r\ x)) \longrightarrow Q\ (\textit{free-fall-flow}\ \tau\ x) \rceil$
  apply(*subst picard-ivp.wp-g-orbit*[*of* $\lambda t\ s.$ *free-fall-kinematics s - - 1 -* ($\lambda\ t\ x.$ *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 $0 \leq t$ **and** $t < 1$
  shows
  $\lceil \lambda s.\ (0{::}real) \leq s\ \$\ (\textit{Abs-three}\ 0) \wedge s\ \$\ (\textit{Abs-three}\ 0) = H \wedge s\ \$\ (\textit{Abs-three}\ 1) = 0 \wedge 0 > s\ \$\ (\textit{Abs-three}\ 2) \rceil$
  $\leq wp\ (\{[x' {=} \lambda t\ s.\ \textit{free-fall-kinematics}\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ (\textit{Abs-three}\ 0) \geq 0)\})$
    $\lceil \lambda s.\ 0 \leq s\ \$\ (\textit{Abs-three}\ 0) \wedge s\ \$\ (\textit{Abs-three}\ 0) \leq H \rceil$
  apply(*subst free-fall-flow-DS*)
  by(*simp-all add*: *assms mult-nonneg-nonpos2*)

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

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-Lindeloef theorem:

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

lemma *picard-ivp-linear-system*:
fixes $A::real\hat{}('n::finite)\hat{}'n$
assumes $0 < ((real\ CARD('n))^2 \cdot (maxAbs\ A))$ (**is** $0 < ?L$)
assumes $0 \le t$ **and** $t < 1/?L$
shows *picard-ivp* $(\lambda\ t\ s.\ A *v\ s)\ \{0..t\}\ UNIV\ ?L\ 0$
apply *unfold-locales* **apply**(*simp add*: ‹$0 \le t$›)
subgoal by(*simp*, *metis continuous-on-compose2 continuous-on-cong continuous-on-id continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest*)
subgoal using *matrix-lipschitz-constant 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* $|r - s| < 1/((real\ CARD('n))^2 \cdot maxAbs\ A)$)
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 $axis\ (1::3)\ (1::real) = (\chi\ j.\ if\ j = 0\ then\ 0\ else\ if\ j = 1\ then\ 1\ else\ 0)$
unfolding *axis-def* **by**(*rule Cart-lambda-cong*, *simp*)

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

abbreviation *flow-for-K t s* ≡ (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-Lindeloef, 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* ≤ *t* ⟹ *t* < *1/9* ⟹
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* ≤ *t* **and** *t* < *1/9*
shows *wp* {[*x*´=λ*t s*. *K* ∗*v s*]{*0*..*t*} *UNIV* @ *0* & *G*} ⌈*Q*⌉ =
⌈λ *x*. ∀ τ ∈ {*0*..*t*}. (∀ *r*∈{*0*−−τ}. *G* (*flow-for-K r x*)) ⟶ *Q* (*flow-for-K* τ *x*)⌉
apply(*subst picard-ivp.wp-g-orbit*[*of* λ*t s*. *K* ∗*v s* - - ((*real CARD*(*3*))^2 · *maxAbs K*) - (λ *t 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* ≤ *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*)})
⌈λ*s*. *0* ≤ *s* $ *0* ∧ *s* $ *0* ≤ *H*⌉
apply(*subst flow-for-K-DS*)
using *assms* **by**(*simp-all add*: *mult-nonneg-nonpos2*)

4.5.3 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*]:$0 \leq x \Longrightarrow 0 > g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \Longrightarrow$ $(x{::}real) \leq H$
proof−
  assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
  then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*
  hence $*$:$v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  from *this* **have** $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*
  also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$
    using *divide-nonneg-neg* **by** *fastforce*
  ultimately have $H - x \geq 0$ **by** *linarith*
  thus *?thesis* **by** *auto*
qed

lemma [*bb-real-arith*]:
  assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
    and *pos*:$g \cdot \tau^2 \ / \ 2 + v \cdot \tau + (x{::}real) = 0$
  shows $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
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 $\lfloor wp\ (IF\ (\lambda s.\ s\ \$\ 0 = 0)\ THEN\ ((\lambda s.\ \eta\ (s[1 :== -\ s\ \$\ 1]))^\bullet)\ ELSE\ \eta^\bullet$
$FI)$
$\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 2 < 0 \wedge 2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 = 2 \cdot s\ \$\ 2 \cdot H + s\ \$\ 1 \cdot s\ \$\ 1\rceil\rfloor$
$= Q$
apply(*subst wp-trafo*) **thm** *wp-trafo*
oops

lemma *bouncing-ball*:
assumes $0 \leq t$ **and** $t < 1/9$
shows $\lceil \lambda s.\ (0{::}real) \leq s\ \$\ (0{::}3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil \leq wp$
$((\{[x'{=}\lambda t\ s.\ K *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)\} \cdot$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ -\ s\ \$\ 1)])\ ELSE\ \eta^\bullet\ FI))^\star)$
$\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H\rceil$
apply(*subst star-nd-fun.abs-eq*, *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*: *fbox-seq*)
apply(*subst p2ndf-ndf2p-wp-sym*[*of* $(IF\ (\lambda s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.$
$-\ s\ \$\ 1)])\ ELSE\ \eta^\bullet\ FI)$])
apply(*subst flow-for-K-DS*) **using** *assms* **apply**(*simp*, *simp*)
oops

4.5.4 Bouncing Ball with invariants

lemma *gravity-is-invariant*:$(x\ solves\text{-}ode\ (\lambda t.\ (\ *v)\ K))\ \{0..t\}\ UNIV \Longrightarrow \tau \in$
$\{0..t\} \Longrightarrow r \in \{0..\tau\} \Longrightarrow$
$((\lambda \tau.\ x\ \tau\ \$\ 2)\ has\text{-}derivative\ (\lambda \tau.\ \tau *_R 0))\ (at\ r\ within\ \{0..\tau\})$
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*:$(x\ solves\text{-}ode\ (\lambda t.\ (\ *v)\ K))\ \{0..t\}\ UNIV \Longrightarrow \tau \in$
$\{0..t\} \Longrightarrow$
$r \in \{0..\tau\} \Longrightarrow ((\lambda \tau.\ 2 \cdot x\ \tau\ \$\ 2 \cdot x\ \tau\ \$\ 0 - 2 \cdot x\ \tau\ \$\ 2 \cdot H - x\ \tau\ \$\ 1 \cdot x\ \tau\ \$$
$1)\ has\text{-}derivative$

($\lambda\tau.\ \tau *_R 0$)) (*at r within* $\{0..\tau\}$)
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*=$\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$
and *g'1*=$\lambda t.\ 2 \cdot (t \cdot (x\ r\ \$\ 1 \cdot x\ r\ \$\ 2))$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac f'1*=$\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$ **and** *g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac f'1*=$\lambda t.\ 0$ **and** *g'1*=($\lambda xa.\ xa \cdot x\ r\ \$\ 1$) **in** *derivative-eq-intros*(*12*))
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*), *simp-all add*: *has-derivative-within-subset*)
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*7*))
apply(*rule-tac g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*), *simp-all add*: *has-derivative-within-subset*)
by(*rule-tac f'1*=($\lambda xa.\ xa \cdot x\ r\ \$\ 2$) **and** *g'1*=($\lambda xa.\ xa \cdot x\ r\ \$\ 2$) **in** *derivative-eq-intros*(*12*),

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

lemma *bouncing-ball-invariants*:
assumes $0 \le t$ **and** $t < 1/9$
shows$\lceil\lambda s.\ (0{::}real) \le s\ \$\ (0{::}3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil \le wp$
$((\{[x' {=} \lambda t\ s.\ K * v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \ge 0)\} \cdot$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ -\ s\ \$\ 1)])\ ELSE\ \eta^\bullet\ FI))^\star)$
$\lceil\lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 \le H\rceil$
apply(*subst star-nd-fun.abs-eq*,
rule-tac I=$\lceil\lambda s.\ 0 \le s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1)\rceil$
in *rel-ad-mka-starI*)
apply(*simp*, *simp only*: *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* ϑ=$\lambda s.\ s\ \$\ 2$ **and** ν=$\lambda s.\ 0$ **in** *dInvariant-below-0*)
using *gravity-is-invariant* **apply** *force*
apply(*simp*, *simp*, *simp*, *simp add*: ‹$0 \le t$›)
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*: ‹$0 \le t$›)
apply(*rule-tac* ϑ=$\lambda s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1$ **and** ν=$\lambda\ s.\ 0$ **in** *dInvariant-eq-0*)
using *bouncing-ball-invariant* **apply** *force*
apply(*simp*, *simp*, *simp*, *simp add*: ‹$0 \le t$›)
apply(*rule dWeakening*, *subst p2ndf-ndf2p-wp*)
oops

4.5.5 Circular motion with invariants

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

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

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

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

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

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

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

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

lemma *flow-for-Circ-solves-Circ*: $((\lambda\ \tau.\ flow\text{-}for\text{-}Circ\ \tau\ s)\ solves\text{-}ode\ (\lambda t\ s.\ \ Circ *v\ s))\ \{0..t\}\ UNIV$
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* $f'1{=}\lambda t.\ -\ s\$0 \cdot (t \cdot sin\ x)$ **and** $g'1{=}\lambda t.\ s\$1 \cdot (t \cdot cos\ x)$**in** *derivative-eq-intros*(*11*))
apply(*rule derivative-eq-intros*(*6*)[*of cos* $(\lambda xa.\ -\ (xa \cdot sin\ x))$]])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of sin* $(\lambda xa.\ (xa \cdot cos\ x))$]])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
subgoal for *x*
apply(*rule-tac* $f'1{=}\lambda t.\ s\$0 \cdot (t \cdot cos\ x)$ **and** $g'1{=}\lambda t.\ -\ s\$1 \cdot (t \cdot sin\ x)$**in** *derivative-eq-intros*(*8*))
apply(*rule derivative-eq-intros*(*6*)[*of sin* $(\lambda xa.\ xa \cdot cos\ x)$]])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of cos* $(\lambda xa.\ -\ (xa \cdot sin\ x))$]])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
done

done

corollary *flow-for-Circ-DS*:
assumes $0 \le t$ **and** $t < 1/4$
shows *wp* $\{[x' {=} \lambda t\ s.\ Circ *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ G\}\ \lceil Q \rceil =$
$\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall\, r {\in} \{0{-}{-}\tau\}.\ G\ (\textit{flow-for-Circ}\ r\ x)) \longrightarrow Q\ (\textit{flow-for-Circ}\ \tau\ x)\rceil$
apply(*subst picard-ivp.wp-g-orbit*[*of* $\lambda t\ s.\ Circ *v\ s$ - - ((*real CARD*(*2*))^2 $\cdot$ *max-Abs Circ*) - (λ *t x. 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 $0 \le t$ **and** $t < 1/4$ **and** $(R{::}real) > 0$
shows$\lceil \lambda s.\ R^2 = (s\ \$\ (0{::}2))^2 + (s\ \$\ 1)^2 \rceil \le wp$
$\{[x' {=} \lambda t\ s.\ Circ *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \ge 0)\}$
$\lceil \lambda s.\ R^2 = (s\ \$\ (0{::}2))^2 + (s\ \$\ 1)^2 \rceil$
apply(*subst flow-for-Circ-DS*)
using *assms* **by** *simp-all*

lemma *circular-motion-invariants*:
assumes $0 \le t$ **and** $t < 1/4$ **and** $(R{::}real) > 0$
shows$\lceil \lambda s.\ R^2 = (s\ \$\ (0{::}2))^2 + (s\ \$\ 1)^2 \rceil \le wp$
$\{[x' {=} \lambda t\ s.\ Circ *v\ s]\{0..t\}\ UNIV\ @\ 0\ \&\ (\lambda\ s.\ s\ \$\ 0 \ge 0)\}$
$\lceil \lambda s.\ R^2 = (s\ \$\ (0{::}2))^2 + (s\ \$\ 1)^2 \rceil$
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*, *clarsimp*)
apply(*erule-tac x=r* **in** *ballE*, *simp-all add*: *matrix-vector-mult-def*)+
sorry
apply(*simp*, *simp*, *simp*, *simp add*: ‹$0 \le t$›) **apply** *auto*[*1*]
by(*rule dWeakening*, *simp*)

end