CPSVerification

CPSVerification

April 20, 2020

Contents

0.1 Hybrid Systems Preliminaries 5
  0.1.1 Functions 5
  0.1.2 Orders 5
  0.1.3 Real numbers 6
  0.1.4 Single variable derivatives 7
  0.1.5 Filters 10
  0.1.6 Multivariable derivatives 11
0.2 Ordinary Differential Equations 13
  0.2.1 Initial value problems and orbits 13
  0.2.2 Differential Invariants 15
  0.2.3 Picard-Lindeloef 20
  0.2.4 Flows for ODEs 22
0.3 Verification components for hybrid systems 28
  0.3.1 Verification of regular programs 28
  0.3.2 Verification of hybrid programs 31
  0.3.3 Derivation of the rules of dL 33
0.4 Mathematical Preliminaries 35
  0.4.1 Syntax 36
  0.4.2 Topology and sets 36
  0.4.3 Functions 37
  0.4.4 Suprema 38
  0.4.5 Real numbers 39
  0.4.6 Vectors and matrices 39
  0.4.7 Diagonalization 41
0.5 Matrix norms 44
  0.5.1 Matrix operator norm 44
  0.5.2 Matrix maximum norm 48
0.6 Square Matrices 50
  0.6.1 Definition 50
  0.6.2 Ring of square matrices 52
  0.6.3 Real normed vector space of square matrices 54
  0.6.4 Real normed algebra of square matrices 56
  0.6.5 Banach space of square matrices 58
  0.6.6 Examples 61

0.7 Affine systems of ODEs 65
0.7.1 Existence and uniqueness for affine systems 65
0.7.2 Flow for affine systems 67
0.7.3 Examples 71
0.8 Verification components with predicate transformers 83
0.8.1 Verification of regular programs 83
0.8.2 Verification of hybrid programs 85
0.8.3 Derivation of the rules of dL 88
0.8.4 Examples 90
0.9 Verification components with Kleene Algebras 98
0.9.1 Hoare logic and refinement in KAT 98
0.9.2 refinement KAT 101
0.9.3 Verification in AKA (KAD) 103
0.9.4 Relational model 104
0.9.5 State transformer model 106
0.10 Verification components with relational MKA 108
0.10.1 Store and weakest preconditions 108
0.10.2 Verification of hybrid programs 109
0.10.3 Derivation of the rules of dL 111
0.11 Verification components with MKA and non-deterministic functions 114
0.11.1 Store and weakest preconditions 115
0.11.2 Verification of hybrid programs 117
0.11.3 Derivation of the rules of dL 119
0.11.4 Examples 122
0.12 Verification and refinement of HS in the relational KAT . . . 129
0.12.1 Store and Hoare triples 130
0.12.2 Verification of hybrid programs 132
0.12.3 Refinement Components 135
0.12.4 Derivation of the rules of dL 138
0.12.5 Examples 140
0.13 Verification and refinement of HS in the relational KAT . . . 153
0.13.1 Store and Hoare triples 153
0.13.2 Verification of hybrid programs 156
0.13.3 Refinement Components 159
0.13.4 Derivation of the rules of dL 162
0.13.5 Examples 163
0.14 Verification components with Kleene Algebras 177
0.14.1 Hoare logic and refinement in KAT 177
0.14.2 refinement KAT 180
0.14.3 Verification in AKA (KAD) 182
0.14.4 Relational model 183
0.14.5 State transformer model 184
0.15 VC_diffKAD 186

0.15.1 Stack Theories Preliminaries: VC_KAD and ODEs . . 187
0.15.2 VC_diffKAD Preliminaries 188
0.15.3 Phase Space Relational Semantics 199
0.15.4 Derivation of Differential Dynamic Logic Rules 202
0.15.5 Rules Testing 219

0.1 Hybrid Systems Preliminaries

Hybrid systems combine continuous dynamics with discrete control. This section contains auxiliary lemmas for verification of hybrid systems.

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

no-notation *has-vderiv-on* (**infix** (*has′-vderiv′-on*) *50*)

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

0.1.1 Functions

lemma *nemptyE*: $T \neq \{\} \Longrightarrow \exists\, t.\ t \in T$
  by *blast*

lemma *funcset-UNIV*: $f \in A \to UNIV$
  by *auto*

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*

0.1.2 Orders

lemma *finite-image-of-finite*[*simp*]:
  fixes f::′*a*::*finite* ⇒ ′*b*
  shows *finite* $\{x.\ \exists\, i.\ x = f\ i\}$
  using *finite-Atleast-Atmost-nat* **by** *force*

lemma *cSup-eq-linorder*:
  fixes *c*::′*a*::*conditionally-complete-linorder*
  assumes $X \neq \{\}$ **and** $\forall\, x \in X.\ x \leq c$
    and *bdd-above* X **and** $\forall\, y{<}c.\ \exists x{\in}X.\ y < x$
  shows $Sup\ X = c$
  by (*meson assms cSup-least less-cSup-iff less-le*)

0.1.3 Real numbers

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

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

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

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

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

lemma *in-real-ivl-eqs*:
$(t \in cball\ t0\ r) = (|t - t0| \leq r)$
$(t \in ball\ t0\ r) = (|t - t0| < r)$
using *dist-real-def* **by** *auto*

lemma *open-cballE*: $t_0 \in T \Longrightarrow open\ T \Longrightarrow \exists\, e>0.\ cball\ t_0\ e \subseteq T$
using *open-contains-cball* **by** *blast*

lemma *open-ballE*: $t_0 \in T \Longrightarrow open\ T \Longrightarrow \exists\, e>0.\ ball\ t_0\ e \subseteq T$
using *open-contains-ball* **by** *blast*

lemma *norm-rotate-simps*[*simp*]:
fixes $x :: 'a{::}\ \{banach,real\text{-}normed\text{-}field\}$
shows $(x * cos\ t - y * sin\ t)^2 + (x * sin\ t + y * cos\ t)^2 = x^2 + y^2$
and $(x * cos\ t + y * sin\ t)^2 + (y * cos\ t - x * sin\ t)^2 = x^2 + y^2$
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*)

thus $(x * cos\ t + y * sin\ t)^2 + (y * cos\ t - x * sin\ t)^2 = x^2 + y^2$
by (*simp add*: *add.commute add.left-commute power2-diff power2-sum*)
qed

0.1.4 Single variable derivatives

— Theorems in the list below are shaped like those on "derivative_eq_intros".

named-theorems *poly-derivatives compilation of optimised miscellaneous derivative rules.*

declare *has-vderiv-on-const* [*poly-derivatives*]
and *has-vderiv-on-id* [*poly-derivatives*]
and *derivative-intros*(*189*) [*poly-derivatives*]
and *derivative-intros*(*190*) [*poly-derivatives*]
and *derivative-intros*(*192*) [*poly-derivatives*]

Below, we consistently name lemmas showing that f' is the derivative of f by starting with "has…". Moreover, if they use the predicate "has_derivative_at", we add them to the list "derivative_intros". Otherwise, if lemmas have an implicit g where $g = f'$, we start their names with "vderiv" and end them with "intro".

lemma *has-derivative-exp-scaleRl*[*derivative-intros*]:
fixes f::*real* ⇒ *real*
assumes $D\ f \mapsto f'$ *at* t *within* T
shows $D\ (\lambda t.\ exp\ (f\ t *_R A)) \mapsto (\lambda h.\ f'\ h *_R (exp\ (f\ t *_R A) * A))$ *at* t *within* T
proof –
from *assms* **have** *bounded-linear* f' **by** *auto*
with *real-bounded-linear* **obtain** m **where** f': $f' = (\lambda h.\ h * m)$ **by** *blast*
show *?thesis*
using *vector-diff-chain-within*[*OF - exp-scaleR-has-vector-derivative-right*, *of f m t T A*]
assms f' **by** (*auto simp*: *has-vector-derivative-def o-def*)
qed

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

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

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

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

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

lemma *has-vderiv-on-power*: $n \geq 1 \Longrightarrow D\ (\lambda t.\ t\ \hat{}\ n) = (\lambda t.\ n * (t\ \hat{}\ (n - 1)))$
$on\ T$
  unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
  by(*rule-tac* $f'1{=}\lambda\ t.\ t$ **in** *derivative-eq-intros*(*16*)) *auto*

lemma *has-vderiv-on-exp*: $D\ (\lambda t.\ exp\ t) = (\lambda t.\ exp\ t)\ on\ T$
 unfolding *has-vderiv-on-def has-vector-derivative-def* **by** (*auto intro*: *derivative-intros*)

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

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

lemma *has-vderiv-on-exp-comp*:
  $D\ (f{::}real \Rightarrow real) = f'\ on\ T \Longrightarrow D\ (\lambda t.\ exp\ (f\ t)) = (\lambda t.\ (f'\ t) * exp\ (f\ t))\ on$
T
  apply(*rule has-vderiv-on-composeI*[*of* $\lambda t.\ exp\ t$])
  by (*rule has-vderiv-on-exp*, *simp-all add*: *mult.commute*)

lemma *has-vderiv-on-exp-scaleRl*:
  assumes $D\ f = f'\ on\ T$
  shows $D\ (\lambda x.\ exp\ (f\ x *_R A)) = (\lambda x.\ f'\ x *_R exp\ (f\ x *_R A) * A)\ on\ T$
 using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
  by (*rule has-derivative-exp-scaleRl*, *auto simp*: *fun-eq-iff*)

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

lemma *vderiv-div-cnstI*[*poly-derivatives*]:
assumes $(a{::}real) \neq 0$ **and** $D\ f = f'\ on\ T$ **and** $g = (\lambda t.\ (f'\ t)/a)$
shows $D\ (\lambda t.\ (f\ t)/a) = g\ on\ T$
apply(*rule has-vderiv-on-composeI*[*of* $\lambda t.\ t/a\ \lambda t.\ 1/a$])
using *assms* **by**(*auto intro*: *has-vderiv-on-divide-cnst*)

lemma *vderiv-npowI*[*poly-derivatives*]:
fixes $f{::}real \Rightarrow real$
assumes $n \geq 1$ **and** $D\ f = f'\ on\ T$ **and** $g = (\lambda t.\ n * (f'\ t) * (f\ t)\hat{}(n-1))$
shows $D\ (\lambda t.\ (f\ t)\hat{}n) = g\ on\ T$
apply(*rule has-vderiv-on-composeI*[*of* $\lambda t.\ t\hat{}n$])
using *assms*(*1*) **apply**(*rule has-vderiv-on-power*)
using *assms* **by** *auto*

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

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

lemma *vderiv-expI*[*poly-derivatives*]:
assumes $D\ (f{::}real \Rightarrow real) = f'\ on\ T$ **and** $g = (\lambda t.\ (f'\ t) * exp\ (f\ t))$
shows $D\ (\lambda t.\ exp\ (f\ t)) = g\ on\ T$
using *assms* **and** *has-vderiv-on-exp-comp* **by** *auto*

lemma *vderiv-on-exp-scaleRlI*[*poly-derivatives*]:
assumes $D\ f = f'\ on\ T$ **and** $g' = (\lambda x.\ f'\ x *_R exp\ (f\ x *_R A) * A)$
shows $D\ (\lambda x.\ exp\ (f\ x *_R A)) = g'\ on\ T$
using *has-vderiv-on-exp-scaleRl assms* **by** *simp*

— Automatically generated derivative rules from this subsection:

thm *derivative-eq-intros*(*140*,*141*,*142*)

— Examples for checking derivatives

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

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

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ a5 * t\hat{}5 + a3 * (t\hat{}3\ /\ c) - a2 * exp\ (t\hat{}2) + a1 * cos\ t + a0) =$
$(\lambda t.\ 5 * a5 * t\hat{}4 + 3 * a3 * (t\hat{}2\ /\ c) - 2 * a2 * t * exp\ (t\hat{}2) - a1 * sin\ t)$ *on* T
by(*auto intro*!: *poly-derivatives*)

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ - a3 * exp\ (t\hat{}3\ /\ c) + a1 * sin\ t + a2 * t\hat{}2) =$
$(\lambda t.\ a1 * cos\ t + 2 * a2 * t - 3 * a3 * t\hat{}2\ /\ c * exp\ (t\hat{}3\ /\ c))$ *on* T
apply(*intro poly-derivatives*)
using *poly-derivatives*(*1*,*2*) **by** *force+*

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c))) =$
$(\lambda t.\ - 4 * a * t\hat{}3 * sin\ (t\hat{}4)\ /\ c * cos\ (cos\ (t\hat{}4)\ /\ c) * exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c)))$ *on* T
apply(*intro poly-derivatives*)
using *poly-derivatives*(*1*,*2*) **by** *force+*

0.1.5 Filters

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

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

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

lemma *eventually-all-finite2*:
fixes P :: ($'a$::*finite*) $\Rightarrow$ $'b$ $\Rightarrow$ *bool*
assumes h:$\forall\, i.$ *eventually* (P i) F
shows *eventually* ($\lambda t.\ \forall\, i.\ P\ i\ t$) F
proof(*unfold eventually-def*)
let *?F* = *Rep-filter* F
have *obs*: $\forall\, i.$ *?F* (P i)
using h **by** *auto*

have *?F* ($\lambda t.\ \forall i \in UNIV.\ P\ i\ t$)
  apply(*rule finite-induct*)
  by(*auto intro*: *eventually-conj simp*: *obs h*)
thus *?F* ($\lambda t.\ \forall i.\ P\ i\ t$)
  by *simp*
qed

lemma *eventually-all-finite-mono*:
  fixes P :: (*'a::finite*) $\Rightarrow$ *'b* $\Rightarrow$ *bool*
  assumes *h1*: $\forall i.$ *eventually* ($P\ i$) F
    and *h2*: $\forall t.\ (\forall i.\ (P\ i\ t)) \longrightarrow Q\ t$
  shows *eventually* $Q\ F$
proof−
  have *eventually* ($\lambda t.\ \forall i.\ P\ i\ t$) F
    using *h1 eventually-all-finite2* **by** *blast*
  thus *eventually* $Q\ F$
    unfolding *eventually-def*
    using *h2 eventually-mono* **by** *auto*
qed

0.1.6 Multivariable derivatives

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

also have $(\{\}::'m\ set) \neq UNIV \wedge finite\ (UNIV :: 'm\ set)$
by *simp*
ultimately have $(\sum i\in UNIV.\ ?c^2 * ((\|?u\ x\ i\|))^2) < (\sum (i::'m)\in UNIV.\ \varepsilon^2 / m)$
by (*metis* (*lifting*) *sum-strict-mono*)
moreover have $?c^2 * (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2) = (\sum i\in UNIV.\ ?c^2 * (\|?u\ x\ i\|)^2)$
using *sum-distrib-left* **by** *blast*
ultimately have $?c^2 * (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2) < \varepsilon^2$
by (*simp add*: *assms*)
hence $sqrt\ (?c^2 * (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2)) < sqrt\ (\varepsilon^2)$
using *real-sqrt-less-iff* **by** *blast*
also have $... = \varepsilon$
using $\langle 0 < \varepsilon\rangle$ **by** *auto*
moreover have $?c * sqrt\ (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2) = sqrt\ (?c^2 * (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2))$
by (*simp add*: *real-sqrt-mult*)
ultimately show $?c * sqrt\ (\sum i\in UNIV.\ (\|?u\ x\ i\|)^2) < \varepsilon$
by *simp*
qed
qed

lemma *tendsto-norm-bound*:
$\forall x.\ \|G\ x - L\| \leq \|F\ x - L\| \Longrightarrow (F \longrightarrow L)\ net \Longrightarrow (G \longrightarrow L)\ net$
apply(*unfold tendsto-iff dist-norm*, *clarsimp*)
apply(*rule-tac* $P{=}\lambda x.\ \|F\ x - L\| < e$ **in** *eventually-mono*, *simp*)
by (*rename-tac e z*) (*erule-tac x=z* **in** *allE*, *simp*)

lemma *tendsto-zero-norm-bound*:
$\forall x.\ \|G\ x\| \leq \|F\ x\| \Longrightarrow (F \longrightarrow 0)\ net \Longrightarrow (G \longrightarrow 0)\ net$
apply(*unfold tendsto-iff dist-norm*, *clarsimp*)
apply(*rule-tac* $P{=}\lambda x.\ \|F\ x\| < e$ **in** *eventually-mono*, *simp*)
by (*rename-tac e z*) (*erule-tac x=z* **in** *allE*, *simp*)

lemma *frechet-vec-nth*:
fixes $f::real \Rightarrow ('a::real\text{-}normed\text{-}vector)\hat{\ }'m$
assumes $((\lambda x.\ (f\ x - f\ x_0 - (x - x_0) *_R f'\ t) /_R (\|x - x_0\|)) \longrightarrow 0)$ (*at t within T*)
shows $((\lambda x.\ (f\ x\ \$\ i - f\ x_0\ \$\ i - (x - x_0) *_R f'\ t\ \$\ i) /_R (\|x - x_0\|)) \longrightarrow 0)$ (*at t within T*)
apply(*rule-tac* $F{=}(\lambda x.\ (f\ x - f\ x_0 - (x - x_0) *_R f'\ t) /_R (\|x - x_0\|))$ **in** *tendsto-zero-norm-bound*)
apply(*clarsimp*, *rule mult-left-mono*)
apply (*metis Finite-Cartesian-Product.norm-nth-le vector-minus-component vector-scaleR-component*)
using *assms* **by** *simp-all*

lemma *has-derivative-vec-lambda*:
fixes $f::real \Rightarrow ('a::banach)\hat{\ }('n::finite)$

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

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

lemma *has-vderiv-on-vec-eq*[*simp*]:
fixes $X{::}real \Rightarrow ('a{::}banach)\hat{}('n{::}finite)$
shows $(D\ X = X'\ on\ T) = (\forall\, i.\ D\ (\lambda t.\ X\ t\ \$\ i) = (\lambda t.\ X'\ t\ \$\ i)\ on\ T)$
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *safe*
using *has-derivative-vec-nth has-derivative-vec-lambda* **by** *blast+*

end

0.2 Ordinary Differential Equations

Vector fields $f{::}real \Rightarrow 'a \Rightarrow ('a{::}real\text{-}normed\text{-}vector)$ represent systems of ordinary differential equations (ODEs). Picard-Lindeloef's theorem guarantees existence and uniqueness of local solutions to initial value problems involving Lipschitz continuous vector fields. A (local) flow $\varphi{::}real \Rightarrow 'a \Rightarrow ('a{::}real\text{-}normed\text{-}vector)$ for such a system is the function that maps initial conditions to their unique solutions. In dynamical systems, the set of all points $\varphi\ t\ s{::}'a$ for a fixed $s{::}'a$ is the flow's orbit. If the orbit of each $s \in I$ is conatined in I, then I is an invariant set of this system. This section formalises these concepts with a focus on hybrid systems (HS) verification.

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

0.2.1 Initial value problems and orbits

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

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

definition *ivp-sols* :: $(real \Rightarrow 'a \Rightarrow ('a{::}real\text{-}normed\text{-}vector)) \Rightarrow real\ set \Rightarrow 'a\ set$
$\Rightarrow$
$real \Rightarrow 'a \Rightarrow (real \Rightarrow 'a)\ set\ (Sols)$

where *Sols* $f\ T\ S\ t_0\ s = \{X \mid X.\ (D\ X = (\lambda t.\ f\ t\ (X\ t))\ on\ T) \wedge X\ t_0 = s \wedge X \in T \rightarrow S\}$

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

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

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

definition *g-orbit* :: $(('a{::}ord) \Rightarrow 'b) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'b\ set\ (\gamma)$
where $\gamma\ X\ G\ T = \bigcup\{\mathcal{P}\ X\ (down\ T\ t) \mid t.\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}\}$

lemma *g-orbit-eq*:
fixes $X{::}('a{::}preorder) \Rightarrow 'b$
shows $\gamma\ X\ G\ T = \{X\ t \mid t.\ t \in T \wedge (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau))\}$
unfolding *g-orbit-def* **apply** *safe*
using *le-left-mono* **by** *blast auto*

lemma $\gamma\ X\ (\lambda s.\ True)\ T = \{X\ t \mid t.\ t \in T\}$ **for** $X{::}('a{::}preorder) \Rightarrow 'b$
unfolding *g-orbit-eq* **by** *simp*

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

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

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

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

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

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

no-notation *g-orbit* (γ)

0.2.2 Differential Invariants

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

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

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

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

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

lemma *diff-invariant-leq-rule* [*diff-invariant-rules*]:
fixes $\mu{::}'a{::}banach \Rightarrow real$
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (\forall \tau{\in}T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ T)$
shows *diff-invariant* $(\lambda s.\ \nu\ s \leq \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix $X\ \tau$ **assume** $\tau \in T$ **and** *x-ivp*:$D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T\ \nu\ (X\ t_0) \leq \mu\ (X\ t_0)$
{assume $\tau \neq t_0$
hence *primed*: $\bigwedge\tau.\ \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall t{\in}T.\ D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\mu'\ (X\ t) - \nu'\ (X\ t)))\ at\ t\ within\ T$
using *assms x-ivp* **by** (*auto simp*: *has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0{<}{-}{-}{<}\tau\} \subseteq T$ $\{t_0{-}{-}\tau\} \subseteq T$
using $\langle\tau \in T\rangle$ *Thyp* $\langle\tau \neq t_0\rangle$ **by** (*auto simp*: *convex-contains-open-segment is-interval-convex-1 closed-segment-subset-interval*)
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ \{t_0{-}{-}\tau\}$
using *obs1 x-ivp* **by** (*auto intro*!: *has-derivative-subset*[*OF - obs2*(*2*)] *simp*: *has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0{<}{-}{-}{<}\tau\}$ **and**
$(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau.\ \tau * (\mu'\ (X\ t) - \nu'\ (X\ t)))\ (\tau - t_0)$
using *mvt-simple-closed-segmentE* $\langle\tau \neq t_0\rangle$ **by** *blast*
hence *mvt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
by *force*
have $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \leq \tau \Longrightarrow t < t_0$ $t \in T$
using $\langle t \in \{t_0{<}{-}{-}{<}\tau\}\rangle$ *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \leq 0$
using *primed*(*1*,*2*)[*OF* $\langle t \in T\rangle$] **by** *auto*
ultimately have $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$
apply(*case-tac* $\tau \geq t_0$)
by (*force*, *auto simp*: *split-mult-pos-le*)
hence $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) \geq 0$
using *x-ivp*(*2*) **by** *auto*
hence $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$
using *mvt* **by** *simp***}**
thus $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$
using *x-ivp* **by** *blast*
qed

lemma *diff-invariant-less-rule* [*diff-invariant-rules*]:

fixes μ::'*a*::*banach* $\Rightarrow$ *real*
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall$ X. $(D\ X = (\lambda\tau.\ f\ (X\ \tau))$ *on* $T) \longrightarrow (\forall\tau{\in}T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))$ *on* $T)$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix X τ **assume** $\tau \in T$ **and** *x-ivp*:$D\ X = (\lambda\tau.\ f\ (X\ \tau))$ *on* T $\nu\ (X\ t_0) < \mu\ (X\ t_0)$
{**assume** $\tau \neq t_0$
hence *primed*: $\bigwedge\tau.\ \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall t{\in}T.\ D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\mu'\ (X\ t) - \nu'\ (X\ t)))$ *at* t *within* T
using *assms x-ivp* **by** (*auto simp*: *has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0{<}{-}{-}{<}\tau\} \subseteq T$ $\{t_0{-}{-}\tau\} \subseteq T$
using ‹$\tau \in T$› *Thyp* ‹$\tau \neq t_0$› **by** (*auto simp*: *convex-contains-open-segment is-interval-convex-1 closed-segment-subset-interval*)
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))$ *on* $\{t_0{-}{-}\tau\}$
using *obs1 x-ivp* **by** (*auto intro*!: *has-derivative-subset*[*OF - obs2*(*2*)] *simp*: *has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0{<}{-}{-}{<}\tau\}$ **and**
$(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau.\ \tau * (\mu'\ (X\ t) - \nu'\ (X\ t)))\ (\tau - t_0)$
using *mvt-simple-closed-segmentE* ‹$\tau \neq t_0$› **by** *blast*
hence *mvt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
by *force*
have $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \leq \tau \Longrightarrow t < t_0$ $t \in T$
using ‹$t \in \{t_0{<}{-}{-}{<}\tau\}$› *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \leq 0$
using *primed*(*1*,*2*)[*OF* ‹$t \in T$›] **by** *auto*
ultimately have $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$
apply(*case-tac* $\tau \geq t_0$)
by (*force*, *auto simp*: *split-mult-pos-le*)
hence $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) > 0$
using *x-ivp*(*2*) **by** *auto*
hence $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
using *mvt* **by** *simp*}
thus $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
using *x-ivp* **by** *blast*
qed

lemma *diff-invariant-nleq-rule*:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
shows *diff-invariant* $(\lambda s.\ \neg\ \nu\ s \leq \mu\ s)\ f\ T\ S\ t_0\ G \longleftrightarrow$ *diff-invariant* $(\lambda s.\ \nu\ s$

```
> μ s) f T S t₀ G
  unfolding diff-invariant-eq
  by safe (clarsimp, erule-tac x=s in allE, simp, erule-tac x=X in ballE, force,
force)+

lemma diff-invariant-neq-rule1 [diff-invariant-rules]:
  fixes μ::'a::banach ⇒ real
  assumes diff-invariant (λs. ν s < μ s) f T S t₀ G
    and diff-invariant (λs. ν s > μ s) f T S t₀ G
  shows diff-invariant (λs. ν s ≠ μ s) f T S t₀ G
proof(unfold diff-invariant-eq, clarsimp)
  fix s::'a and X::real ⇒ 'a and t::real
  assume ν s ≠ μ s and Xhyp: X ∈ Sols (λt. f) T S t₀ s
     and thyp: t ∈ T and Ghyp: ∀τ. τ ∈ T ∧ τ ≤ t ⟶ G (X τ)
  hence ν s < μ s ∨ ν s > μ s
    by linarith
  moreover have ν s < μ s ⟹ ν (X t) < μ (X t)
    using assms(1) Xhyp thyp Ghyp unfolding diff-invariant-eq by auto
  moreover have ν s > μ s ⟹ ν (X t) > μ (X t)
    using assms(2) Xhyp thyp Ghyp unfolding diff-invariant-eq by auto
  ultimately show ν (X t) = μ (X t) ⟹ False
    by auto
qed

lemma IVT-two-functions:
  fixes f :: ('a::{linear-continuum-topology, real-vector}) ⇒
  ('b::{linorder-topology,real-normed-vector,ordered-ab-group-add})
  assumes conts: continuous-on {a..b} f continuous-on {a..b} g
      and ahyp: f a < g a and bhyp: g b < f b  and a ≤ b
    shows ∃x∈{a..b}. f x = g x
proof−
  let ?h x = f x − g x
  have ?h a ≤ 0 and ?h b ≥ 0
    using ahyp bhyp by simp-all
  also have continuous-on {a..b} ?h
    using conts continuous-on-diff by blast
  ultimately obtain x where a ≤ x x ≤ b and ?h x = 0
    using IVT'[of ?h] ‹a ≤ b› by blast
  thus ?thesis
    using ‹a ≤ b› by auto
qed

lemma IVT-two-functions-real-ivl:
  fixes f :: real ⇒ real
  assumes conts: continuous-on {a−−b} f continuous-on {a−−b} g
      and ahyp: f a < g a and bhyp: g b < f b
    shows ∃x∈{a−−b}. f x = g x
proof(cases a ≤ b)
  case True
```

then show *?thesis*
using *IVT-two-functions assms*
unfolding *closed-segment-eq-real-ivl* **by** *auto*
next
case *False*
hence $a \geq b$
by *auto*
hence *continuous-on* $\{b..a\}$ *f continuous-on* $\{b..a\}$ *g*
using *conts False* **unfolding** *closed-segment-eq-real-ivl* **by** *auto*
hence $\exists\, x \in \{b..a\}.\ g\ x = f\ x$
using *IVT-two-functions*[*of b a g f*] *assms*(*3*,*4*) *False* **by** *auto*
then show *?thesis*
using ‹$a \geq b$› **unfolding** *closed-segment-eq-real-ivl* **by** *auto force*
qed

lemma *diff-invariant-neq-rule2* [*diff-invariant-rules*]:
fixes μ::*'a*::*banach* ⇒ *real*
assumes *Thyp*: *is-interval* T $t_0 \in T$ $\forall\, t \in T.\ t_0 \leq t$
and *conts*: $\forall\, X.$ ($D\ X = (\lambda\tau.\ f\ (X\ \tau))$ *on* T) $\longrightarrow$ *continuous-on* ($\mathcal{P}\ X\ T$) ν
$\wedge$ *continuous-on* ($\mathcal{P}\ X\ T$) μ
and *dIhyp*:*diff-invariant* ($\lambda s.\ \nu\ s \neq \mu\ s$) $f\ T\ S\ t_0\ G$
shows *diff-invariant* ($\lambda s.\ \nu\ s < \mu\ s$) $f\ T\ S\ t_0\ G$
proof(*unfold diff-invariant-eq*, *clarsimp*)
fix s::*'a* **and** X::*real* ⇒ *'a* **and** t::*real*
assume *ineq0*: $\nu\ s < \mu\ s$ **and** *Xhyp*: $X \in$ *Sols* ($\lambda t.\ f$) $T\ S\ t_0\ s$
and *Ghyp*: $\forall\, \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$ **and** *thyp*: $t \in T$
hence *ineq1*: $\nu\ (X\ t_0) < \mu\ (X\ t_0)$
by (*auto simp*: *ivp-solsD*)
have $t_0 \leq t$ **and** $\mu\ (X\ t) \neq \nu\ (X\ t)$
using ‹$t \in T$› *Thyp dIhyp thyp Xhyp Ghyp ineq0* **unfolding** *diff-invariant-eq*
by *force+*
moreover
{**assume** *ineq2*:$\nu\ (X\ t) > \mu\ (X\ t)$
note *continuous-on-compose*[*OF vderiv-on-continuous-on*[*OF ivp-solsD*(*1*)[*OF*
Xhyp]]]
hence *continuous-on* T ($\nu \circ X$) **and** *continuous-on* T ($\mu \circ X$)
using *ivp-solsD*(*1*)[*OF Xhyp*] *conts* **by** *auto*
also have $\{t_0{-}{-}t\} \subseteq T$
using *Thyp thyp* **by** (*simp add*: *closed-segment-subset-interval*)
ultimately have *continuous-on* $\{t_0{-}{-}t\}$ ($\lambda\tau.\ \nu\ (X\ \tau)$) **and** *continuous-on*
$\{t_0{-}{-}t\}$ ($\lambda\tau.\ \mu\ (X\ \tau)$)
using *continuous-on-subset* **by** *auto*
then obtain τ **where** $\tau \in \{t_0{-}{-}t\}$ $\mu\ (X\ \tau) = \nu\ (X\ \tau)$
using *IVT-two-functions-real-ivl*[*OF - - ineq1 ineq2*] **by** *force*
hence $\forall\, r \in$ *down* $T\ \tau.\ G\ (X\ r)$ **and** $\tau \in T$
using *Ghyp* ‹$\tau \in \{t_0{-}{-}t\}$› ‹$t_0 \leq t$› ‹$\{t_0{-}{-}t\} \subseteq T$› **by** (*auto simp*: *closed-segment-eq-real-ivl*)
hence $\mu\ (X\ \tau) \neq \nu\ (X\ \tau)$
using *dIhyp Xhyp* ‹$\nu\ s < \mu\ s$› **unfolding** *diff-invariant-eq* **by** *force*
hence *False*

using ‹μ (X τ) = ν (X τ)› **by** *blast*}
ultimately show ν (X t) < μ (X t)
by *fastforce*
qed

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

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

0.2.3 Picard-Lindeloef

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

locale *picard-lindeloef* =
fixes f::*real* $\Rightarrow$ ($'a$::{*heine-borel*,*banach*}) $\Rightarrow$ $'a$ **and** T::*real set* **and** S::$'a$ *set*
and t_0::*real*
assumes *open-domain*: *open* T *open* S
and *interval-time*: *is-interval* T
and *init-time*: $t_0 \in T$
and *cont-vec-field*: $\forall s \in S.$ *continuous-on* T ($\lambda t.$ f t s)
and *lipschitz-vec-field*: *local-lipschitz* T S f
begin

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

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

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

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

lemma *unique-solution*:
assumes *xivp*: D X = ($\lambda t.$ f t (X t)) *on* {t_0−−t} X t_0 = s $X \in$ {t_0−−t} $\rightarrow$ S
and $t \in T$
and *yivp*: D Y = ($\lambda t.$ f t (Y t)) *on* {t_0−−t} Y t_0 = s $Y \in$ {t_0−−t} $\rightarrow$ S **and**
$s \in S$
shows X t = Y t

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

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

end

lemma *local-lipschitz-add*:
  fixes *f1 f2* :: *real* ⇒ *'a*::*banach* ⇒ *'a*
  assumes *local-lipschitz T S f1*
     and *local-lipschitz T S f2*
    shows *local-lipschitz T S* ($\lambda t\ s.\ f1\ t\ s + f2\ t\ s$)
proof(*unfold local-lipschitz-def*, *clarsimp*)
  fix *s* **and** *t* **assume** $s \in S$ **and** $t \in T$
  obtain ε_1 *L1* **where** $\varepsilon_1 > 0$ **and** *L1*: $\bigwedge \tau.\ \tau{\in}cball\ t\ \varepsilon_1 \cap T \Longrightarrow L1{-}lipschitz\text{-}on$
$(cball\ s\ \varepsilon_1 \cap S)\ (f1\ \tau)$
    using *local-lipschitzE*[*OF assms*(*1*) ‹$t \in T$› ‹$s \in S$›] **by** *blast*

obtain ε_2 *L2* **where** $\varepsilon_2 > 0$ **and** *L2*: $\bigwedge\tau.\ \tau{\in}cball\ t\ \varepsilon_2 \cap T \Longrightarrow L2{-}lipschitz\text{-}on$ $(cball\ s\ \varepsilon_2 \cap S)\ (f2\ \tau)$
using *local-lipschitzE*[*OF assms*(*2*) ‹$t \in T$› ‹$s \in S$›] **by** *blast*
have *ballH*: $cball\ s\ (min\ \varepsilon_1\ \varepsilon_2) \cap S \subseteq cball\ s\ \varepsilon_1 \cap S$ $cball\ s\ (min\ \varepsilon_1\ \varepsilon_2) \cap S \subseteq cball\ s\ \varepsilon_2 \cap S$
by *auto*
have *obs1*: $\forall\tau{\in}cball\ t\ \varepsilon_1 \cap T.\ L1{-}lipschitz\text{-}on\ (cball\ s\ (min\ \varepsilon_1\ \varepsilon_2) \cap S)\ (f1\ \tau)$
using *lipschitz-on-subset*[*OF L1 ballH*(*1*)] **by** *blast*
also have *obs2*: $\forall\tau{\in}cball\ t\ \varepsilon_2 \cap T.\ L2{-}lipschitz\text{-}on\ (cball\ s\ (min\ \varepsilon_1\ \varepsilon_2) \cap S)$ $(f2\ \tau)$
using *lipschitz-on-subset*[*OF L2 ballH*(*2*)] **by** *blast*
ultimately have $\forall\tau{\in}cball\ t\ (min\ \varepsilon_1\ \varepsilon_2) \cap T.$
$(L1 + L2){-}lipschitz\text{-}on\ (cball\ s\ (min\ \varepsilon_1\ \varepsilon_2) \cap S)\ (\lambda s.\ f1\ \tau\ s + f2\ \tau\ s)$
using *lipschitz-on-add* **by** *fastforce*
thus $\exists u{>}0.\ \exists L.\ \forall t{\in}cball\ t\ u \cap T.\ L{-}lipschitz\text{-}on\ (cball\ s\ u \cap S)\ (\lambda s.\ f1\ t\ s +$ $f2\ t\ s)$
apply(*rule-tac* $x{=}min\ \varepsilon_1\ \varepsilon_2$ **in** *exI*)
using ‹$\varepsilon_1 > 0$› ‹$\varepsilon_2 > 0$› **by** *force*
qed

lemma *picard-lindeloef-add*: $picard\text{-}lindeloef\ f1\ T\ S\ t_0 \Longrightarrow picard\text{-}lindeloef\ f2\ T\ S$ $t_0 \Longrightarrow$
$picard\text{-}lindeloef\ (\lambda t\ s.\ f1\ t\ s + f2\ t\ s)\ T\ S\ t_0$
unfolding *picard-lindeloef-def* **apply**(*clarsimp*, *rule conjI*)
using *continuous-on-add* **apply** *fastforce*
using *local-lipschitz-add* **by** *blast*

lemma *picard-lindeloef-constant*: $picard\text{-}lindeloef\ (\lambda t\ s.\ c)\ UNIV\ UNIV\ t_0$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
by (*rule-tac* $x{=}1$ **in** *exI*, *clarsimp*, *rule-tac* $x{=}1/2$ **in** *exI*, *simp*)

0.2.4 Flows for ODEs

A locale designed for verification of hybrid systems. The user can select the interval of existence and the defining flow equation via the variables T and φ.

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

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

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

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

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

lemma *has-derivative-on-open1*:
assumes $t > 0\ t \in T\ s \in S$
obtains B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ *at t within B*
proof−
obtain *r*::*real* **where** *rHyp*: $r > 0$ *ball* $t\ r \subseteq T$
using *open-contains-ball-eq open-domain*(*1*) ‹$t \in T$› **by** *blast*
moreover have $t + r/2 > 0$
using ‹$r > 0$› ‹$t > 0$› **by** *auto*
moreover have $\{0--t\} \subseteq T$
using *subintervalI*[*OF init-time* ‹$t \in T$›] **.**
ultimately have *subs*: $\{0<--<t + r/2\} \subseteq T$
unfolding *abs-le-eq abs-le-eq real-ivl-eqs*[*OF* ‹$t > 0$›] *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›]
by *clarify* (*case-tac* $t < x$, *simp-all add*: *cball-def ball-def dist-norm subset-eq field-simps*)
have $t + r/2 \in T$
using *rHyp* **unfolding** *real-ivl-eqs*[*OF rHyp*(*1*)] **by** (*simp add*: *subset-eq*)
hence $\{0--t + r/2\} \subseteq T$
using *subintervalI*[*OF init-time*] **by** *blast*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))$ *on* $\{0--(t + r/2)\})$
using *ivp*(*1*)[*OF* - ‹$s \in S$›] **by** *auto*
hence *vderiv*: $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))$ *on* $\{0<--<t + r/2\})$
apply(*rule has-vderiv-on-subset*)
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **by** *auto*
have $t \in \{0<--<t + r/2\}$
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **using** *rHyp* ‹$t > 0$› **by** *simp*
moreover have $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ (*at t within* $\{0<--<t + r/2\}$)
using *vderiv calculation* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *blast*
moreover have *open* $\{0<--<t + r/2\}$
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **by** *simp*

ultimately show *?thesis*
using *subs that* **by** *blast*
qed

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

lemma *has-derivative-on-open3*:
assumes $s \in S$
obtains B **where** $0 \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ 0\ s))$ *at 0 within B*
proof−
obtain r::*real* **where** *rHyp*: $r > 0$ *ball 0 r* $\subseteq T$
using *open-contains-ball-eq open-domain*(*1*) *init-time* **by** *blast*
hence $r/2 \in T$ $-r/2 \in T$ $r/2 > 0$
unfolding *real-ivl-eqs* **by** *auto*
hence *subs*: $\{0{-}{-}r/2\} \subseteq T$ $\{0{-}{-}(-r/2)\} \subseteq T$

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

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

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

lemma *has-vderiv-on-domain*:
assumes $s \in S$
shows $D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ T$

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

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

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

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

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

lemma *is-monoid-action*:
  assumes $s \in S$ **and** $T = UNIV$

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

definition *orbit* :: $'a \Rightarrow 'a\ set$ (γ^φ)
where $\gamma^\varphi\ s$ = *g-orbital* f $(\lambda s.\ True)$ $T\ S\ 0\ s$

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

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

by *blast*
qed

end

lemma *line-is-local-flow*:
$0 \in T \Longrightarrow$ *is-interval* $T \Longrightarrow$ *open* $T \Longrightarrow$ *local-flow* $(\lambda\ s.\ c)$ T *UNIV* $(\lambda\ t\ s.\ s + t *_R c)$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac* $x=1$ **in** *exI*, *clarsimp*, *rule-tac* $x=1/2$ **in** *exI*, *simp*)
apply(*rule-tac* $f'1=\lambda\ s.\ 0$ **and** $g'1=\lambda\ s.\ c$ **in** *derivative-intros*(*189*))
apply(*rule derivative-intros*, *simp*)+
by *simp-all*

end

0.3 Verification components for hybrid systems

A light-weight version of the verification components. We define the forward box operator to compute weakest liberal preconditions (wlps) of hybrid programs. Then we introduce three methods for verifying correctness specifications of the continuous dynamics of a HS.

theory *hs-vc-spartan*
imports *hs-prelims-dyn-sys*

begin

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

no-notation *Transitive-Closure.rtrancl* ((-*) [*1000*] *999*)

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

abbreviation *skip* $\equiv (\lambda s.\ \{s\})$

0.3.1 Verification of regular programs

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

definition *fbox* :: $('a \Rightarrow 'b\ set) \Rightarrow 'b\ pred \Rightarrow 'a\ pred$ (|-] - [*61*,*81*] *82*)
where $|F]\ P = (\lambda s.\ (\forall s'.\ s' \in F\ s \longrightarrow P\ s'))$

lemma *fbox-iso*: $P \leq Q \Longrightarrow |F]\ P \leq |F]\ Q$
unfolding *fbox-def* **by** *auto*

lemma *fbox-invariants*:
assumes $I \leq |F]\ I$ **and** $J \leq |F]\ J$

shows $(\lambda s.\ I\ s \wedge J\ s) \leq |F]\ (\lambda s.\ I\ s \wedge J\ s)$
and $(\lambda s.\ I\ s \vee J\ s) \leq |F]\ (\lambda s.\ I\ s \vee J\ s)$
using *assms* **unfolding** *fbox-def* **by** *auto*

Now, we compute wlps for specific programs.

lemma *fbox-eta*[*simp*]: *fbox skip* $P = P$
unfolding *fbox-def* **by** *simp*

Next, we introduce assignments and their wlps.

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

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

lemma *fbox-assign*[*simp*]: $|x ::= e]\ Q = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j))$
unfolding *vec-upd-def assign-def* **by** (*subst fbox-def*) *simp*

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

definition *kcomp* :: $('a \Rightarrow 'b\ set) \Rightarrow ('b \Rightarrow 'c\ set) \Rightarrow ('a \Rightarrow 'c\ set)$ (**infixl** ; *75*)
where
$F\ ;\ G = \mu \circ \mathcal{P}\ G \circ F$

lemma *kcomp-eq*: $(f\ ;\ g)\ x = \bigcup\ \{g\ y\ |y.\ y \in f\ x\}$
unfolding *kcomp-def image-def* **by** *auto*

lemma *fbox-kcomp*[*simp*]: $|G\ ;\ F]\ P = |G]\ |F]\ P$
unfolding *fbox-def kcomp-def* **by** *auto*

lemma *fbox-kcomp-ge*:
assumes $P \leq |G]\ R\ R \leq |F]\ Q$
shows $P \leq |G\ ;\ F]\ Q$
apply(*subst fbox-kcomp*)
by (*rule order.trans*[*OF assms*(*1*)]) (*rule fbox-iso*[*OF assms*(*2*)])

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

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

lemma *fbox-if-then-else*[*simp*]:
$|IF\ T\ THEN\ X\ ELSE\ Y]\ Q = (\lambda s.\ (T\ s \longrightarrow (\ |X]\ Q)\ s) \wedge (\neg\ T\ s \longrightarrow (\ |Y]\ Q)\ s))$
unfolding *fbox-def ifthenelse-def* **by** *auto*

lemma *hoare-if-then-else*:
assumes $(\lambda s.\ P\ s \wedge T\ s) \leq |X]\ Q$

```
  and (λs. P s ∧ ¬ T s) ≤ |Y] Q
 shows P ≤ |IF T THEN X ELSE Y] Q
 using assms unfolding fbox-def ifthenelse-def by auto
```

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

```
definition kpower :: ('a ⇒ 'a set) ⇒ nat ⇒ ('a ⇒ 'a set)
 where kpower f n = (λs. ((;) f ^^ n) skip s)

lemma kpower-base:
 shows kpower f 0 s = {s} and kpower f (Suc 0) s = f s
 unfolding kpower-def by(auto simp: kcomp-eq)

lemma kpower-simp: kpower f (Suc n) s = (f ; kpower f n) s
 unfolding kcomp-eq apply(induct n)
 unfolding kpower-base apply(rule subset-antisym, clarsimp, force, clarsimp)
 unfolding kpower-def kcomp-eq by simp

definition kleene-star :: ('a ⇒ 'a set) ⇒ ('a ⇒ 'a set) ((-*) [1000] 999)
 where (f*) s = ⋃ {kpower f n s |n. n ∈ UNIV}

lemma kpower-inv:
 fixes F :: 'a ⇒ 'a set
 assumes ∀ s. I s ⟶ (∀ s'. s' ∈ F s ⟶ I s')
 shows ∀ s. I s ⟶ (∀ s'. s' ∈ (kpower F n s) ⟶ I s')
 apply(clarsimp, induct n)
 unfolding kpower-base kpower-simp apply(simp-all add: kcomp-eq, clarsimp)
 apply(subgoal-tac I y, simp)
 using assms by blast

lemma kstar-inv: I ≤ |F] I ⟹ I ≤ |F*] I
 unfolding kleene-star-def fbox-def apply clarsimp
 apply(unfold le-fun-def, subgoal-tac ∀ x. I x ⟶ (∀ s'. s' ∈ F x ⟶ I s'))
 using kpower-inv[of I F] by blast simp

lemma fbox-kstarI:
 assumes P ≤ I and I ≤ Q and I ≤ |F] I
 shows P ≤ |F*] Q
proof−
 have I ≤ |F*] I
  using assms(3) kstar-inv by blast
 hence P ≤ |F*] I
  using assms(1) by auto
 also have |F*] I ≤ |F*] Q
  by (rule fbox-iso[OF assms(2)])
 finally show ?thesis .
qed

definition loopi :: ('a ⇒ 'a set) ⇒ 'a pred ⇒ ('a ⇒ 'a set) (LOOP - INV -
[64,64] 63)
```

where *LOOP F INV I* $\equiv$ (F^*)

lemma *fbox-loopI*: $P \leq I \Longrightarrow I \leq Q \Longrightarrow I \leq |F]\ I \Longrightarrow P \leq |LOOP\ F\ INV\ I]\ Q$
unfolding *loopi-def* **using** *fbox-kstarI*[*of P*] **by** *simp*

0.3.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: $(('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b\ pred \Rightarrow 'a\ set \Rightarrow ('b \Rightarrow 'b\ set)$ $(EVOL)$
where $EVOL\ \varphi\ G\ T = (\lambda s.\ g\text{-}orbit\ (\lambda t.\ \varphi\ t\ s)\ G\ T)$

lemma *fbox-g-evol*[*simp*]:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $|EVOL\ \varphi\ G\ T]\ Q = (\lambda s.\ (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
unfolding *g-evol-def g-orbit-eq fbox-def* **by** *auto*

Verification by providing solutions

lemma *fbox-g-orbital*: $|x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0]\ Q =$
$(\lambda s.\ \forall X{\in}Sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
unfolding *fbox-def g-orbital-eq* **by** (*auto simp*: *fun-eq-iff*)

context *local-flow*
begin

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

lemma *fbox-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow |x'{=}f\ \&\ G\ on\ \{0..t\}\ S\ @\ 0]\ Q =$
$(\lambda s.\ s \in S \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
unfolding *fbox-g-orbital* **apply**(*rule ext*, *clarsimp*, *safe*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ s$ **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *ivp-sols-def*)
apply(*subgoal-tac* $\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ X\ \tau = \varphi\ \tau\ s)$, *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add*: *ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp*: *closed-segment-eq-real-ivl*)
apply(*force simp*: *closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

lemma *fbox-orbit*: $|\gamma^{\varphi}]\ Q = (\lambda s.\ s \in S \longrightarrow (\forall\ t \in T.\ Q\ (\varphi\ t\ s)))$

unfolding *orbit-def fbox-g-ode* **by** *simp*

end

Verification with differential invariants

definition *g-ode-inv* :: $(('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow 'a\ set)$ $((1x\acute{}{=}\text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}\ DINV\ \text{-}\ ))$
where $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$

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

lemma *fbox-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq |x\acute{}=f\ \&\ G\ on\ T\ S\ @\ t_0]\ I$ **and** $I \leq Q$
shows $P \leq |x\acute{}=f\ \&\ G\ on\ T\ S\ @\ t_0]\ Q$
using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
by (*rule fbox-iso*[*OF assms*(*3*)])

lemma *fbox-diff-inv*[*simp*]:
$(I \leq |x\acute{}=f\ \&\ G\ on\ T\ S\ @\ t_0]\ I) =$ *diff-invariant* $I\ f\ T\ S\ t_0\ G$
by (*auto simp*: *diff-invariant-def ivp-sols-def fbox-def g-orbital-eq*)

lemma *diff-inv-guard-ignore*:
assumes $I \leq |x\acute{}= f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0]\ I$
shows $I \leq |x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ I$
using *assms* **unfolding** *fbox-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*
begin

lemma *fbox-diff-inv-eq*: *diff-invariant* $I\ f\ T\ S\ 0\ (\lambda s.\ True) =$
$((\lambda s.\ s \in S \longrightarrow I\ s) = |x\acute{}= f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ 0]\ (\lambda s.\ s \in S \longrightarrow I\ s))$
unfolding *fbox-diff-inv*[*symmetric*] *fbox-g-orbital le-fun-def fun-eq-iff*
using *init-time* **apply**(*clarsimp simp*: *subset-eq ivp-sols-def*)
apply(*safe*, *force*, *force*)
apply(*subst ivp*(*2*)[*symmetric*], *simp*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *allE*)
using *in-domain has-vderiv-on-domain ivp*(*2*) *init-time* **by** *auto*

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

end

lemma *fbox-g-odei*: $P \leq I \Longrightarrow I \leq |x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ I \Longrightarrow (\lambda s.\ I\ s \wedge G$

s) $\leq$ *Q* $\Longrightarrow$
  P $\leq$ |*x*´= *f* & *G on T S* @ t_0 *DINV I*] *Q*
   unfolding *g-ode-inv-def* **apply**(*rule-tac b*=|*x*´= *f* & *G on T S* @ t_0] *I* **in** *order.trans*)
   apply(*rule-tac I*=*I* **in** *fbox-g-orbital-inv*, *simp-all*)
  apply(*subst fbox-g-orbital-guard*, *simp*)
  by (*rule fbox-iso*, *force*)

0.3.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

lemma *diff-solve-axiom*:
  fixes *c*::′*a*::{*heine-borel*, *banach*}
  assumes *0* $\in$ *T* **and** *is-interval T open T*
  shows |*x*´=(λ*s*. *c*) & *G on T UNIV* @ *0*] *Q* =
  (λ*s*. $\forall$ *t*$\in$*T*. ($\mathcal{P}$ ($\lambda\tau$. *s* + τ $*_R$ *c*) (*down T t*) $\subseteq$ {*s*. *G s*}) $\longrightarrow$ *Q* (*s* + *t* $*_R$ *c*))
  apply(*subst local-flow.fbox-g-ode*[*of* λ*s*. *c* - - (λ*t s*. *s* + *t* $*_R$ *c*)])
  using *line-is-local-flow assms* **by** *auto*

lemma *diff-solve-rule*:
  assumes *local-flow f T UNIV* φ
    and $\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\in$*T*. ($\mathcal{P}$ (λ*t*. φ *t s*) (*down T t*) $\subseteq$ {*s*. *G s*}) $\longrightarrow$ *Q* (φ *t s*))
  shows *P* $\leq$ |*x*´= *f* & *G on T UNIV* @ *0*] *Q*
  using *assms* **by**(*subst local-flow.fbox-g-ode*) *auto*

lemma *diff-weak-axiom*: |*x*´= *f* & *G on T S* @ t_0] *Q* = |*x*´= *f* & *G on T S* @ t_0] (λ*s*. *G s* $\longrightarrow$ *Q s*)
  unfolding *fbox-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*: *G* $\leq$ *Q* $\Longrightarrow$ *P* $\leq$ |*x*´= *f* & *G on T S* @ t_0] *Q*
  by(*auto intro*: *g-orbitalD simp*: *le-fun-def g-orbital-eq fbox-def*)

lemma *fbox-g-orbital-eq-univD*:
  assumes |*x*´= *f* & *G on T S* @ t_0] *C* = (λ*s*. *True*)
    and $\forall\tau\in$(*down T t*). *x* τ $\in$ (*x*´= *f* & *G on T S* @ t_0) *s*
  shows $\forall\tau\in$(*down T t*). *C* (*x* τ)
proof
  fix τ **assume** τ $\in$ (*down T t*)
  hence *x* τ $\in$ (*x*´= *f* & *G on T S* @ t_0) *s*
    using *assms*(*2*) **by** *blast*
  also have $\forall$ *s*′. *s*′ $\in$ (*x*´= *f* & *G on T S* @ t_0) *s* $\longrightarrow$ *C s*′
    using *assms*(*1*) **unfolding** *fbox-def* **by** *meson*
  ultimately show *C* (*x* τ) **by** *blast*
qed

lemma *diff-cut-axiom*:

assumes *Thyp*: *is-interval* T $t_0 \in T$
and $|x\acute{} = f$ & G *on* T S @ $t_0]$ $C = (\lambda s.$ *True*)
shows $|x\acute{} = f$ & G *on* T S @ $t_0]$ $Q = |x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0]$ Q
proof(*rule-tac* $f{=}\lambda\ x.\ |x]\ Q$ **in** *HOL.arg-cong*, *rule ext*, *rule subset-antisym*)
fix s
{**fix** s' **assume** $s' \in (x\acute{} = f$ & G *on* T S @ $t_0)$ s
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in$ *Sols* $(\lambda t.\ f)$ T S t_0 s
and $X\ \tau = s'$ **and** $\tau \in T$ **and** *guard-x*:$\mathcal{P}$ X (*down* T τ) $\subseteq \{s.\ G\ s\}$
using *g-orbitalD*[*of* s' f G T S t_0 s] **by** *blast*
have $\forall\, t\in$(*down* T τ). $\mathcal{P}$ X (*down* T t) $\subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall\, t\in$(*down* T τ). $t \in T$
using ‹$\tau \in T$› *Thyp closed-segment-subset-interval* **by** *auto*
ultimately have $\forall\, t\in$(*down* T τ). $X\ t \in (x\acute{} = f$ & G *on* T S @ $t_0)$ s
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence $\forall\, t\in$(*down* T τ). C $(X\ t)$
using *assms*(*3*) **unfolding** *fbox-def* **by** *meson*
hence $s' \in (x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0)$ s
using *g-orbitalI*[*OF x-ivp* ‹$\tau \in T$›] *guard-x* ‹$X\ \tau = s'$› **by** *fastforce*}
thus $(x\acute{} = f$ & G *on* T S @ $t_0)$ $s \subseteq (x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0)$ s
by *blast*
next show $\bigwedge s.\ (x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0)$ $s \subseteq (x\acute{} = f$ & G *on* T S @ $t_0)$ s
by (*auto simp*: *g-orbital-eq*)
qed

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and *fbox-C*: $P \leq |x\acute{} = f$ & G *on* T S @ $t_0]$ C
and *fbox-Q*: $P \leq |x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0]$ Q
shows $P \leq |x\acute{} = f$ & G *on* T S @ $t_0]$ Q
proof(*subst fbox-def*, *subst g-orbital-eq*, *clarsimp*)
fix t::*real* **and** X::*real* $\Rightarrow$ *'a* **and** s **assume** $P\ s$ **and** $t \in T$
and *x-ivp*:$X \in$ *Sols* $(\lambda t.\ f)$ T S t_0 s
and *guard-x*:$\forall\,\tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
have $\forall\,\tau\in$(*down* T t). $X\ \tau \in (x\acute{} = f$ & G *on* T S @ $t_0)$ s
using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
hence $\forall\,\tau\in$(*down* T t). C $(X\ \tau)$
using *fbox-C* ‹$P\ s$› **by** (*subst* (*asm*) *fbox-def*, *auto*)
hence $X\ t \in (x\acute{} = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* T S @ $t_0)$ s
using *guard-x* ‹$t \in T$› **by** (*auto intro*!: *g-orbitalI x-ivp*)
thus Q $(X\ t)$
using ‹$P\ s$› *fbox-Q* **by** (*subst* (*asm*) *fbox-def*) *auto*
qed

The rules of dL

abbreviation *g-global-orbit* ::((*'a*::*banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'a* $\Rightarrow$ *'a set*
((*1x´=- & -*)) **where** $(x\acute{}{=}f$ & $G) \equiv (x\acute{}{=}f$ & G *on UNIV UNIV* @ *0*)

abbreviation *g-global-ode-inv* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a ⇒ 'a set
((1x´=- & - DINV -)) **where** (x´=f & G DINV I) ≡ (x´=f & G on UNIV UNIV @ 0 DINV I)

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

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

lemma *DW*: |x´= f & G] Q = |x´= f & G] (λs. G s ⟶ Q s)
by (*rule diff-weak-axiom*)

lemma *dW*: G ≤ Q ⟹ P ≤ |x´= f & G] Q
by (*rule diff-weak-rule*)

lemma *DC*:
assumes |x´= f & G] C = (λs. *True*)
shows |x´= f & G] Q = |x´= f & (λs. G s ∧ C s)] Q
by (*rule diff-cut-axiom*) (*auto simp*: *assms*)

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

lemma *dI*:
assumes P ≤ I **and** *diff-invariant I f UNIV UNIV 0 G* **and** I ≤ Q
shows P ≤ |x´= f & G] Q
by (*rule fbox-g-orbital-inv*[*OF assms*(*1*) - *assms*(*3*)]) (*simp add*: *assms*(*2*))

end

0.4 Mathematical Preliminaries

This section adds useful syntax, abbreviations and theorems to the Isabelle distribution.

theory *mtx-prelims*
imports *hs-prelims*

begin

0.4.1 Syntax

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

syntax
-ivl-integral :: *real* $\Rightarrow$ *real* $\Rightarrow$ *'a* $\Rightarrow$ *pttrn* $\Rightarrow$ *bool* $((3\int$ -⁻ (-)∂/-) [0, 0, 10] 10)

translations
$\int_a{}^b f\ \partial x \rightleftharpoons$ *CONST ivl-integral a b* $(\lambda x.\ f)$

notation *matrix-inv* $(\text{-}^{-1}\ [90])$

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

0.4.2 Topology and sets

lemmas *compact-imp-bdd-above* = *compact-imp-bounded*[*THEN bounded-imp-bdd-above*]

lemma *comp-cont-image-spec*: *continuous-on T f* $\Longrightarrow$ *compact T* $\Longrightarrow$ *compact* $\{f\ t \mid t.\ t \in T\}$
using *compact-continuous-image* **by** (*simp add*: *Setcompr-eq-image*)

lemmas *bdd-above-cont-comp-spec* = *compact-imp-bdd-above*[*OF comp-cont-image-spec*]

lemmas *bdd-above-norm-cont-comp* = *continuous-on-norm*[*THEN bdd-above-cont-comp-spec*]

lemma *cSup-norm-cont-comp-ge*:
$t \in T \Longrightarrow$ *continuous-on T f* $\Longrightarrow$ *compact T* $\Longrightarrow \|f\ t\| \leq Sup\ \{\|f\ t\| \mid t.\ t \in T\}$
by (*rule cSup-upper*[*OF - bdd-above-norm-cont-comp*], *auto*)

lemma *cSup-norm-cont-comp-ge0*:
$T \neq \{\} \Longrightarrow$ *continuous-on T f* $\Longrightarrow$ *compact T* $\Longrightarrow 0 \leq Sup\ \{\|f\ t\| \mid t.\ t \in T\}$
apply(*drule nemptyE*, *clarsimp*)
subgoal for *t*
by(*rule order.trans* [*OF - cSup-norm-cont-comp-ge*[*of t*]], *auto*) **.**

lemma *open-cballE*: $t_0 \in T \Longrightarrow$ *open T* $\Longrightarrow \exists\, e{>}0.\ cball\ t_0\ e \subseteq T$
using *open-contains-cball* **by** *blast*

lemma *open-ballE*: $t_0 \in T \Longrightarrow$ *open T* $\Longrightarrow \exists\, e{>}0.\ ball\ t_0\ e \subseteq T$
using *open-contains-ball* **by** *blast*

lemma *funcset-UNIV*: $f \in A \rightarrow UNIV$
by *auto*

lemma *finite-image-of-finite*[*simp*]:
  fixes f::$'a$::*finite* $\Rightarrow$ $'b$
  shows *finite* $\{x.\ \exists\, i.\ x = f\ i\}$
  using *finite-Atleast-Atmost-nat* **by** *force*

lemma *finite-image-of-finite2*:
  fixes f :: $'a$::*finite* $\Rightarrow$ $'b$::*finite* $\Rightarrow$ $'c$
  shows *finite* $\{f\ x\ y\ |x\ y.\ P\ x\ y\}$
proof−
  have *finite* $(\bigcup x.\ \{f\ x\ y|y.\ P\ x\ y\})$
    by *simp*
  moreover have $\{f\ x\ y|x\ y.\ P\ x\ y\} = (\bigcup x.\ \{f\ x\ y|y.\ P\ x\ y\})$
    by *auto*
  ultimately show *?thesis*
    by *simp*
qed

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

0.4.3 Functions

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

lemma *suminfI*:
  fixes f :: *nat* $\Rightarrow$ $'a$::$\{$*t2-space*,*comm-monoid-add*$\}$
  shows f *sums* $k \Longrightarrow$ *suminf* $f = k$
  unfolding *sums-iff* **by** *simp*

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

lemma *suminf-multr*: *summable* $f \Longrightarrow (\sum n.\ f\ n * c) = (\sum n.\ f\ n) * c$ **for** c::$'a$::*real-normed-algebra*
  by (*rule bounded-linear.suminf* [*OF bounded-linear-mult-left*, *symmetric*])

lemma *sum-if-then-else-simps*[*simp*]:
  fixes q :: ($'a$::*semiring-0*) **and** i :: $'n$::*finite*
  shows $(\sum j{\in}UNIV.\ f\ j * (\mathit{if}\ j = i\ \mathit{then}\ q\ \mathit{else}\ 0)) = f\ i * q$

and $(\sum j{\in}UNIV.\ f\ j * (if\ i = j\ then\ q\ else\ 0)) = f\ i * q$
and $(\sum j{\in}UNIV.\ (if\ i = j\ then\ q\ else\ 0) * f\ j) = q * f\ i$
and $(\sum j{\in}UNIV.\ (if\ j = i\ then\ q\ else\ 0) * f\ j) = q * f\ i$
by (*auto simp*: *finite-sum-univ-singleton*[*of* - *i*])

0.4.4 Suprema

lemma *le-max-image-of-finite*[*simp*]:
fixes $f::'a::finite \Rightarrow 'b::linorder$
shows $(f\ i) \le Max\ \{x.\ \exists\, i.\ x = f\ i\}$
by (*rule Max.coboundedI*, *simp-all*) (*rule-tac x=i* **in** *exI*, *simp*)

lemma *cSup-eq*:
fixes $c::'a::conditionally\text{-}complete\text{-}lattice$
assumes $\forall\, x \in X.\ x \le c$ **and** $\exists\, x \in X.\ c \le x$
shows $Sup\ X = c$
by (*metis assms cSup-eq-maximum order-class.order.antisym*)

lemma *cSup-mem-eq*:
$c \in X \Longrightarrow \forall\, x \in X.\ x \le c \Longrightarrow Sup\ X = c$ **for** $c::'a::conditionally\text{-}complete\text{-}lattice$
by (*rule cSup-eq*, *auto*)

lemma *cSup-finite-ex*:
$finite\ X \Longrightarrow X \neq \{\} \Longrightarrow \exists\, x{\in}X.\ Sup\ X = x$ **for** $X::'a::conditionally\text{-}complete\text{-}linorder$ *set*
by (*metis* (*full-types*) *bdd-finite*(*1*) *cSup-upper finite-Sup-less-iff order-less-le*)

lemma *cMax-finite-ex*:
$finite\ X \Longrightarrow X \neq \{\} \Longrightarrow \exists\, x{\in}X.\ Max\ X = x$ **for** $X::'a::conditionally\text{-}complete\text{-}linorder$ *set*
apply(*subst cSup-eq-Max*[*symmetric*])
using *cSup-finite-ex* **by** *auto*

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

using *Max-ge* **by** *blast*
hence $n \geq$ *?bound i*
using $\langle n \geq$ *?N*$\rangle$ **by** *linarith*
hence *P i n*
using *obs* **by** *blast*}
thus $\exists N.\ \forall i\ n.\ N \leq n \longrightarrow P\ i\ n$
by *blast*
qed

0.4.5 Real numbers

named-theorems *field-power-simps simplification rules for powers to the nth*

declare *semiring-normalization-rules(18)* [*field-power-simps*]
and *semiring-normalization-rules(26)* [*field-power-simps*]
and *semiring-normalization-rules(27)* [*field-power-simps*]
and *semiring-normalization-rules(28)* [*field-power-simps*]
and *semiring-normalization-rules(29)* [*field-power-simps*]

WARNING: Adding $?x * ?x^{?q} = ?x^{Suc\ ?q}$ to our tactic makes its combination with simp to loop infinitely in some proofs.

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

lemma *frac-diff-eq1*: $a \neq b \Longrightarrow a\ /\ (a - b) - b\ /\ (a - b) = 1$ **for** *a*::*real*
by (*metis* (*no-types*, *hide-lams*) *ab-left-minus add.commute add-left-cancel diff-divide-distrib diff-minus-eq-add div-self*)

lemma *exp-add*: $x * y - y * x = 0 \Longrightarrow exp\ (x + y) = exp\ x * exp\ y$
by (*rule exp-add-commuting*) (*simp add*: *ac-simps*)

lemmas *mult-exp-exp* = *exp-add*[*symmetric*]

0.4.6 Vectors and matrices

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

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

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

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

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

lemma $uminus\text{-}axis\text{-}eq[simp]$: $-\ axis\ i\ k = axis\ i\ (-k)$ **for** $k :: 'a::ring$
  unfolding $axis\text{-}def$ **by**$(simp\ add\text{: } vec\text{-}eq\text{-}iff)$

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

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

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

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

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

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

0.4.7 Diagonalization

lemma *invertibleI*: $A ** B = mat\ 1 \Longrightarrow B ** A = mat\ 1 \Longrightarrow invertible\ A$
  unfolding *invertible-def* **by** *auto*

lemma *invertibleD*[*simp*]:
  assumes *invertible A*
  shows $A^{-1} ** A = mat\ 1$ **and** $A ** A^{-1} = mat\ 1$
  using *assms* **unfolding** *matrix-inv-def invertible-def*
  by (*simp-all add*: *verit-sko-ex'*)

lemma *matrix-inv-unique*:
  assumes $A ** B = mat\ 1$ **and** $B ** A = mat\ 1$
  shows $A^{-1} = B$
  by (*metis assms invertibleD*(*2*) *invertibleI matrix-mul-assoc matrix-mul-lid*)

lemma *invertible-matrix-inv*: $invertible\ A \Longrightarrow invertible\ (A^{-1})$
  using *invertibleD invertibleI* **by** *blast*

lemma *matrix-inv-idempotent*[*simp*]: $invertible\ A \Longrightarrow A^{-1-1} = A$
  using *invertibleD matrix-inv-unique* **by** *blast*

lemma *matrix-inv-matrix-mul*:
  assumes *invertible A* **and** *invertible B*
  shows $(A ** B)^{-1} = B^{-1} ** A^{-1}$
proof(*rule matrix-inv-unique*)
  have $A ** B ** (B^{-1} ** A^{-1}) = A ** (B ** B^{-1}) ** A^{-1}$
    by (*simp add*: *matrix-mul-assoc*)
  also have ... $= mat\ 1$
    using *assms* **by** *simp*
  finally show $A ** B ** (B^{-1} ** A^{-1}) = mat\ 1$ **.**
next
  have $B^{-1} ** A^{-1} ** (A ** B) = B^{-1} ** (A^{-1} ** A) ** B$
    by (*simp add*: *matrix-mul-assoc*)
  also have ... $= mat\ 1$

using *assms* **by** *simp*
finally show $B^{-1} ** A^{-1} ** (A ** B) = mat\ 1$ **.**
qed

lemma *mat-inverse-simps*[*simp*]:
fixes c :: *'a::division-ring*
assumes $c \neq 0$
shows *mat* (*inverse c*) $**$ *mat c* = *mat 1*
and *mat c* $**$ *mat* (*inverse c*) = *mat 1*
unfolding *matrix-matrix-mult-def mat-def* **by** (*auto simp*: *vec-eq-iff assms*)

lemma *matrix-inv-mat*[*simp*]: $c \neq 0 \Longrightarrow (mat\ c)^{-1} = mat\ (inverse\ c)$ **for** c :: *'a::division-ring*
by (*simp add*: *matrix-inv-unique*)

lemma *invertible-mat*[*simp*]: $c \neq 0 \Longrightarrow$ *invertible* (*mat c*) **for** c :: *'a::division-ring*
using *invertibleI mat-inverse-simps*(*1*) *mat-inverse-simps*(*2*) **by** *blast*

lemma *matrix-inv-mat-1*: $(mat\ (1::'a::division\text{-}ring))^{-1} = mat\ 1$
by *simp*

lemma *invertible-mat-1*: *invertible* (*mat* (*1::'a::division-ring*))
by *simp*

definition *similar-matrix* :: (*'a::semiring-1*) ^*'m*^*'m* ⇒ (*'a::semiring-1*) ^*'n*^*'n* ⇒ *bool* (**infixr** ∼ *25*)
where *similar-matrix A B* ⟷ $(\exists\ P.\ invertible\ P \wedge A = P^{-1} ** B ** P)$

lemma *similar-matrix-refl*[*simp*]: $A \sim A$ **for** A :: *'a::division-ring*^*'n*^*'n*
by (*unfold similar-matrix-def*, *rule-tac x=mat 1* **in** *exI*, *simp*)

lemma *similar-matrix-simm*: $A \sim B \Longrightarrow B \sim A$ **for** $A\ B$ :: (*'a::semiring-1*) ^*'n*^*'n*
apply(*unfold similar-matrix-def*, *clarsimp*)
apply(*rule-tac* $x=P^{-1}$ **in** *exI*, *simp add*: *invertible-matrix-inv*)
by (*metis invertible-def matrix-inv-unique matrix-mul-assoc matrix-mul-lid matrix-mul-rid*)

lemma *similar-matrix-trans*: $A \sim B \Longrightarrow B \sim C \Longrightarrow A \sim C$ **for** $A\ B\ C$ :: (*'a::semiring-1*) ^*'n*^*'n*
proof(*unfold similar-matrix-def*, *clarsimp*)
fix $P\ Q$
assume $A = P^{-1} ** (Q^{-1} ** C ** Q) ** P$ **and** $B = Q^{-1} ** C ** Q$
let $?R = Q ** P$
assume *inverts*: *invertible Q invertible P*
hence $?R^{-1} = P^{-1} ** Q^{-1}$
by (*rule matrix-inv-matrix-mul*)
also have *invertible ?R*
using *inverts invertible-mult* **by** *blast*
ultimately show $\exists R.\ invertible\ R \wedge P^{-1} ** (Q^{-1} ** C ** Q) ** P = R^{-1} ** C ** R$

  by (*metis matrix-mul-assoc*)
qed

lemma *mat-vec-nth-simps*[*simp*]:
 $i = j \Longrightarrow mat\ c\ \$\ i\ \$\ j = c$
 $i \neq j \Longrightarrow mat\ c\ \$\ i\ \$\ j = 0$
 by (*simp-all add*: *mat-def*)

definition *diag-mat f* = (χ *i j*. *if i = j then f i else 0*)

lemma *diag-mat-vec-nth-simps*[*simp*]:
 $i = j \Longrightarrow diag\text{-}mat\ f\ \$\ i\ \$\ j = f\ i$
 $i \neq j \Longrightarrow diag\text{-}mat\ f\ \$\ i\ \$\ j = 0$
 unfolding *diag-mat-def* **by** *simp-all*

lemma *diag-mat-const-eq*[*simp*]: *diag-mat* (λi. *c*) = *mat c*
 unfolding *mat-def diag-mat-def* **by** *simp*

lemma *matrix-vector-mul-diag-mat*: *diag-mat f* $*v$ *s* = (χ *i*. *f i* $*$ *s*\$*i*)
 unfolding *diag-mat-def matrix-vector-mult-def* **by** *simp*

lemma *matrix-vector-mul-diag-axis*[*simp*]: *diag-mat f* $*v$ (*axis i k*) = *axis i* (*f i* $*$ *k*)
 by (*simp add*: *matrix-vector-mul-diag-mat axis-def fun-eq-iff*)

lemma *matrix-mul-diag-matl*: *diag-mat f* $**$ *A* = (χ *i j*. *f i* $*$ *A*\$*i*\$*j*)
 unfolding *diag-mat-def matrix-matrix-mult-def* **by** *simp*

lemma *matrix-matrix-mul-diag-matr*: *A* $**$ *diag-mat f* = (χ *i j*. *A*\$*i*\$*j* $*$ *f j*)
 unfolding *diag-mat-def matrix-matrix-mult-def* **apply**(*clarsimp simp*: *fun-eq-iff*)
 subgoal for *i j*
  by (*auto simp*: *finite-sum-univ-singleton*[*of - j*])
 done

lemma *matrix-mul-diag-diag*: *diag-mat f* $**$ *diag-mat g* = *diag-mat* (λi. *f i* $*$ *g i*)
 unfolding *diag-mat-def matrix-matrix-mult-def vec-eq-iff* **by** *simp*

lemma *compow-matrix-mul-diag-mat-eq*: (($**$) (*diag-mat f*) ^^ *n*) (*mat 1*) = *diag-mat* (λi. *f i*^*n*)
 apply(*induct n*, *simp-all add*: *matrix-mul-diag-matl*)
 by (*auto simp*: *vec-eq-iff diag-mat-def*)

lemma *compow-similar-diag-mat-eq*:
 assumes *invertible P*
   and $A = P^{-1}$ $**$ (*diag-mat f*) $**$ *P*
  shows (($**$) *A* ^^ *n*) (*mat 1*) = P^{-1} $**$ (*diag-mat* (λi. *f i*^*n*)) $**$ *P*
proof(*induct n*, *simp-all add*: *assms*)
 fix *n*::*nat*
 have P^{-1} $**$ *diag-mat f* $**$ *P* $**$ (P^{-1} $**$ *diag-mat* (λi. *f i* ^ *n*) $**$ *P*) =

P^{-1} ** *diag-mat f* ** *diag-mat* ($\lambda i.$ *f i* ^ *n*) ** *P* (**is** *?lhs* = -)
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *invertibleD*(*2*) *matrix-mul-rid matrix-mul-assoc*)

also have ... = P^{-1} ** *diag-mat* ($\lambda i.$ *f i* * *f i* ^ *n*) ** *P* (**is** - = *?rhs*)
by (*metis* (*full-types*) *matrix-mul-assoc matrix-mul-diag-diag*)
finally show *?lhs* = *?rhs* **.**
qed

lemma *compow-similar-diag-mat*:
assumes *A* $\sim$ (*diag-mat f*)
shows ((**) *A* ^^ *n*) (*mat 1*) $\sim$ *diag-mat* ($\lambda i.$ *f i*^*n*)
proof(*unfold similar-matrix-def*)
obtain *P* **where** *invertible P* **and** *A* = P^{-1} ** (*diag-mat f*) ** *P*
using *assms* **unfolding** *similar-matrix-def* **by** *blast*
thus $\exists$ *P*. *invertible P* $\wedge$ ((**) *A* ^^ *n*) (*mat 1*) = P^{-1} ** *diag-mat* ($\lambda i.$ *f i* ^ *n*) ** *P*
using *compow-similar-diag-mat-eq* **by** *blast*
qed

no-notation *matrix-inv* ($-^{-1}$ [*90*])
and *similar-matrix* (**infixr** $\sim$ *25*)

end

0.5 Matrix norms

Here, we explore some properties about the operator and the maximum norms for matrices.

theory *mtx-norms*
imports *mtx-prelims*

begin

0.5.1 Matrix operator norm

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

lemma *norm-matrix-bound*:
fixes *A* :: (*'a::real-normed-algebra-1*)^*'n*^*'m*
shows $\|x\|$ = *1* $\Longrightarrow$ $\|$*A* *v *x*$\|$ $\leq$ $\|$(χ *i j*. $\|$*A* $ *i* $ *j*$\|$) *v *1*$\|$
proof−
fix *x* :: (*'a*, *'n*) *vec* **assume** $\|x\|$ = *1*
hence *xi-le1*:$\bigwedge$*i*. $\|$*x* $ *i*$\|$ $\leq$ *1*
by (*metis Finite-Cartesian-Product.norm-nth-le*)
{**fix** *j*::*'m*

```
    have ‖(∑ i∈UNIV. A $ j $ i * x $ i)‖ ≤ (∑ i∈UNIV. ‖A $ j $ i * x $ i‖)
      using norm-sum by blast
    also have ... ≤ (∑ i∈UNIV. (‖A $ j $ i‖) * (‖x $ i‖))
      by (simp add: norm-mult-ineq sum-mono)
    also have ... ≤ (∑ i∈UNIV. (‖A $ j $ i‖) * 1)
      using xi-le1 by (simp add: sum-mono mult-left-le)
    finally have ‖(∑ i∈UNIV. A $ j $ i * x $ i)‖ ≤ (∑ i∈UNIV. (‖A $ j $ i‖)
* 1) by simp}
  hence ⋀j. ‖(A *v x) $ j‖ ≤ ((χ i1 i2. ‖A $ i1 $ i2‖) *v 1) $ j
    unfolding matrix-vector-mult-def by simp
  hence (∑ j∈UNIV. (‖(A *v x) $ j‖)^2) ≤ (∑ j∈UNIV. (‖((χ i1 i2. ‖A $ i1 $
i2‖) *v 1) $ j‖)^2)
   by (metis (mono-tags, lifting) norm-ge-zero power2-abs power-mono real-norm-def
sum-mono)
  thus ‖A *v x‖ ≤ ‖(χ i j. ‖A $ i $ j‖) *v 1‖
    unfolding norm-vec-def L2-set-def by simp
qed

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

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

lemma op-norm-def: ‖A‖_op = Sup {‖A *v x‖ | x. ‖x‖ = 1}
  apply(rule antisym[OF onorm-le cSup-least[OF op-norm-set-proptys(3)]])
   apply(case-tac x = 0, simp)
   apply(subst mult-norm-matrix-sgn-eq[symmetric], simp)
   apply(rule cSup-upper[OF - op-norm-set-proptys(2)])
   apply(force simp: norm-sgn)
  unfolding onorm-def apply(rule cSup-upper[OF - onorm-set-proptys(2)])
  by (simp add: image-def, clarsimp) (metis div-by-1)
```

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

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

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

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

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

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

lemma *op-norm0*: $\|(0::('a::real\text{-}normed\text{-}field)\,\hat{}\,'n\,\hat{}\,'m)\|_{op} = 0$
using *op-norm-eq-0*[*of 0*] **by** *simp*

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

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

lemma *op-norm-matrix-matrix-mult-le*: $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$
proof(*rule onorm-le*)
have $0 \leq (\|A\|_{op})$
by(*rule onorm-pos-le*[*OF blin-matrix-vector-mult*])
fix *x* **have** $\|A ** B *v\ x\| = \|A *v\ (B *v\ x)\|$
by (*simp add*: *matrix-vector-mul-assoc*)
also have ... $\leq (\|A\|_{op}) * (\|B *v\ x\|)$
by (*simp add*: *norm-matrix-le-mult-op-norm*[*of - B *v x*])
also have ... $\leq (\|A\|_{op}) * ((\|B\|_{op}) * (\|x\|))$

using *norm-matrix-le-mult-op-norm*[*of B x*] ‹$0 \leq (\|A\|_{op})$› *mult-left-mono* **by** *blast*
finally show $\|A ** B *v\ x\| \leq (\|A\|_{op}) * (\|B\|_{op}) * (\|x\|)$
by *simp*
qed

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

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

lemma *op-norm-le-transpose*: $\|A\|_{op} \leq \|transpose\ A\|_{op}$ **for** $A :: real\hat{}'n\hat{}'n$
proof−
have *obs*:$\forall x.\ \|x\| = 1 \longrightarrow (\|A *v\ x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
using *norm-matrix-vec-mult-le-transpose* **by** *blast*
have $(\|A\|_{op}) \leq sqrt\ ((\|transpose\ A ** A\|_{op}))$
using *obs* **apply**(*unfold op-norm-def*)

by (*rule cSup-least*[*OF op-norm-set-proptys*(*3*)]) *clarsimp*
hence $((\|A\|_{op}))^2 \le (\|transpose\ A ** A\|_{op})$
using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *op-norm-ge-0* **by** (*metis not-le real-less-lsqrt*)
also have ... $\le (\|transpose\ A\|_{op}) * (\|A\|_{op})$
using *op-norm-matrix-matrix-mult-le* **by** *blast*
finally have $((\|A\|_{op}))^2 \le (\|transpose\ A\|_{op}) * (\|A\|_{op})$ **by** *linarith*
thus $(\|A\|_{op}) \le (\|transpose\ A\|_{op})$
using *sq-le-cancel*[*of* $(\|A\|_{op})$] *op-norm-ge-0* **by** *metis*
qed

0.5.2 Matrix maximum norm

abbreviation *max-norm* :: *real*^'*n*^'*m* ⇒ *real* ((*1*$\|$-$\|_{max}$) [*65*] *61*)
where $\|A\|_{max} \equiv$ *Max* (*abs* ' (*entries A*))

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

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

lemma *max-norm-ge-0*: $0 \le \|A\|_{max}$
unfolding *max-norm-def*
apply(*rule order.trans*[*OF abs-ge-zero*[*of A* \$ - \$ -] *Max-ge*])
using *max-norm-set-proptys* **by** *auto*

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

lemma *sqrt-Sup-power2-eq-Sup-abs*:
finite A ⟹ *A* ≠ {} ⟹ *sqrt* (*Sup* $\{(f\ i)^2\ |i.\ i \in A\}$) = *Sup* $\{|f\ i|\ |i.\ i \in A\}$
proof(*rule sym*)
assume *assms*: *finite A A* ≠ {}
then obtain *i* **where** *i-def*: $i \in A \wedge Sup\ \{(f\ i)^2|i.\ i \in A\} = (f\ i)$^*2*
using *cSup-finite-ex*[*of* $\{(f\ i)^2|i.\ i \in A\}$] **by** *auto*
hence *lhs*: *sqrt* (*Sup* $\{(f\ i)^2\ |i.\ i \in A\}$) = $|f\ i|$
by *simp*

have *finite* $\{(f\ i)^2|i.\ i \in A\}$
  using *assms* **by** *simp*
hence $\forall j{\in}A.\ (f\ j)^2 \leq (f\ i)^2$
  using *i-def cSup-upper*[*of* - $\{(f\ i)^2\ |i.\ i \in A\}$] **by** *force*
hence $\forall j{\in}A.\ |f\ j| \leq |f\ i|$
  using *abs-le-square-iff* **by** *blast*
also have $|f\ i| \in \{|f\ i|\ |i.\ i \in A\}$
  using *i-def* **by** *auto*
ultimately show $Sup\ \{|f\ i|\ |i.\ i \in A\} = sqrt\ (Sup\ \{(f\ i)^2\ |i.\ i \in A\})$
  using *cSup-mem-eq*[*of* $|f\ i|$ $\{|f\ i|\ |i.\ i \in A\}$] *lhs* **by** *auto*
qed

lemma *sqrt-Max-power2-eq-max-abs*:
  $finite\ A \Longrightarrow A \neq \{\} \Longrightarrow sqrt\ (Max\ \{(f\ i)^2|i.\ i \in A\}) = Max\ \{|f\ i|\ |i.\ i \in A\}$
  apply(*subst cSup-eq-Max*[*symmetric*], *simp-all*)+
  using *sqrt-Sup-power2-eq-Sup-abs* **.**

lemma *op-norm-diag-mat-eq*: $\|diag\text{-}mat\ f\|_{op} = Max\ \{|f\ i|\ |i.\ i \in UNIV\}$ (**is** - = *Max ?A*)
proof(*unfold op-norm-def*)
  have *obs*: $\bigwedge x\ i.\ (f\ i)^2 * (x\ \$\ i)^2 \leq Max\ \{(f\ i)^2|i.\ i \in UNIV\} * (x\ \$\ i)^2$
    apply(*rule mult-right-mono*[*OF - zero-le-power2*])
    using *le-max-image-of-finite*[*of* $\lambda i.\ (f\ i)\hat{}2$] **by** *simp*
  {**fix** *r* **assume** $r \in \{\|diag\text{-}mat\ f *v\ x\|\ |x.\ \|x\| = 1\}$
    then obtain *x* **where** *x-def*: $\|diag\text{-}mat\ f *v\ x\| = r \wedge \|x\| = 1$
      by *blast*
    hence $r^2 = (\sum i{\in}UNIV.\ (f\ i)^2 * (x\ \$\ i)^2)$
      unfolding *norm-vec-def L2-set-def matrix-vector-mul-diag-mat*
      apply (*simp add*: *power-mult-distrib*)
     by (*metis* (*no-types*, *lifting*) *x-def norm-ge-zero real-sqrt-ge-0-iff real-sqrt-pow2*)
    also have ... $\leq (Max\ \{(f\ i)^2|i.\ i \in UNIV\}) * (\sum i{\in}UNIV.\ (x\ \$\ i)^2)$
      using *obs*[*of* - *x*] **by** (*simp add*: *sum-mono sum-distrib-left*)
    also have ... $= Max\ \{(f\ i)^2|i.\ i \in UNIV\}$
      using *x-def* **by** (*simp add*: *norm-vec-def L2-set-def*)
    finally have $r \leq sqrt\ (Max\ \{(f\ i)^2|i.\ i \in UNIV\})$
      using *x-def real-le-rsqrt* **by** *blast*
    hence $r \leq Max\ ?A$
      by (*subst* (*asm*) *sqrt-Max-power2-eq-max-abs*[*of UNIV f*], *simp-all*)}
  hence *1*: $\forall x{\in}\{\|diag\text{-}mat\ f *v\ x\|\ |x.\ \|x\| = 1\}.\ x \leq Max\ ?A$
    unfolding *diag-mat-def* **by** *blast*
  obtain *i* **where** *i-def*: $Max\ ?A = \|diag\text{-}mat\ f *v\ \mathrm{e}\ i\|$
    using *cMax-finite-ex*[*of ?A*] **by** *force*
  hence *2*: $\exists x{\in}\{\|diag\text{-}mat\ f *v\ x\|\ |x.\ \|x\| = 1\}.\ Max\ ?A \leq x$
    by (*metis* (*mono-tags*, *lifting*) *abs-1 mem-Collect-eq norm-axis-eq order-refl real-norm-def*)
  show $Sup\ \{\|diag\text{-}mat\ f *v\ x\|\ |x.\ \|x\| = 1\} = Max\ ?A$
    by (*rule cSup-eq*[*OF 1 2*])
qed

lemma *op-max-norms-eq-at-diag*: $\|diag\text{-}mat\ f\|_{op} = \|diag\text{-}mat\ f\|_{max}$
proof(*rule antisym*)
  have $\{|f\ i|\ |i.\ i \in UNIV\} \subseteq \{|diag\text{-}mat\ f\ \$\ i\ \$\ j|\ |i\ j.\ i \in UNIV \wedge j \in UNIV\}$
    by (*smt Collect-mono diag-mat-vec-nth-simps*(*1*))
  thus $\|diag\text{-}mat\ f\|_{op} \leq \|diag\text{-}mat\ f\|_{max}$
    unfolding *op-norm-diag-mat-eq max-norm-def*
    by (*rule Max.subset-imp*) (*blast*, *simp only*: *finite-image-of-finite2*)
next
  have $Sup\ \{|diag\text{-}mat\ f\ \$\ i\ \$\ j|\ |i\ j.\ i \in UNIV \wedge j \in UNIV\} \leq Sup\ \{|f\ i|\ |i.\ i \in UNIV\}$
    apply(*rule cSup-least*, *blast*, *clarify*, *case-tac* $i = j$, *simp*)
    by (*rule cSup-upper*, *blast*, *simp-all*) (*rule cSup-upper2*, *auto*)
  thus $\|diag\text{-}mat\ f\|_{max} \leq \|diag\text{-}mat\ f\|_{op}$
    unfolding *op-norm-diag-mat-eq max-norm-def*
    apply (*subst cSup-eq-Max*[*symmetric*], *simp only*: *finite-image-of-finite2*, *blast*)
    by (*subst cSup-eq-Max*[*symmetric*], *simp*, *blast*)
qed

end

0.6 Square Matrices

The general solution for affine systems of ODEs involves the exponential function. Unfortunately, this operation is only available in Isabelle for the type class "banach". Hence, we define a type of square matrices and prove that it is an instance of this class.

theory *mtx-sq*
  imports *mtx-norms*

begin

0.6.1 Definition

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

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

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

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

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

lift-definition *vec-sq-mtx-prod* :: (*real^'m*) ⇒ *'m sq-mtx* ⇒ (*real^'m*) **is** (*v*$*$*) .

lift-definition *sq-mtx-diag* :: $(('m{::}finite) \Rightarrow real) \Rightarrow ('m{::}finite)$ *sq-mtx* (**binder** diag *10*) **is** *diag-mat* **.**

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

lift-definition *sq-mtx-inv* :: $('m{::}finite)$ *sq-mtx* $\Rightarrow$ $'m$ *sq-mtx* (-^{-1} [*90*]) **is** *matrix-inv* **.**

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

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

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

lemma *to-mtx-ith*[*simp*]:
  (*to-mtx A*) \$\$ *i1* = *A* \$ *i1*
  (*to-mtx A*) \$\$ *i1* \$ *i2* = *A* \$ *i1* \$ *i2*
  by (*transfer*, *simp*)+

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

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

lemma *sq-mtx-diag-simps*[*simp*]:
  i = *j* $\Longrightarrow$ *sq-mtx-diag f* \$\$ *i* \$ *j* = *f i*
  i $\neq$ *j* $\Longrightarrow$ *sq-mtx-diag f* \$\$ *i* \$ *j* = *0*
  sq-mtx-diag f \$\$ *i* = *axis i* (*f i*)
  unfolding *sq-mtx-diag-def* **by** (*simp-all add*: *axis-def vec-eq-iff*)

lemma *sq-mtx-diag-vec-mult*: (diag *i*. *f i*) $*_V$ *s* = (χ *i*. *f i* $*$ *s*\$*i*)
 by (*simp add*: *matrix-vector-mul-diag-mat sq-mtx-diag.abs-eq sq-mtx-vec-mult.abs-eq*)

lemma *sq-mtx-vec-mult-diag-axis*: (diag *i*. *f i*) $*_V$ (*axis i k*) = *axis i* (*f i* $*$ *k*)
  unfolding *sq-mtx-diag-vec-mult axis-def* **by** *auto*

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

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

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

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

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

0.6.2 Ring of square matrices

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

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

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

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

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

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

declare *plus-sq-mtx.rep-eq* [*simp*]
and *minus-sq-mtx.rep-eq* [*simp*]

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

end

lemma *sq-mtx-zero-ith*[*simp*]: *0* $$ i = *0*
by (*transfer*, *simp*)

lemma *sq-mtx-zero-nth*[*simp*]: *0* $$ i $ j = *0*
by *transfer simp*

lemma *sq-mtx-plus-eq*: $A + B$ = *to-mtx* (χ i j. $A$$$i$$j$ + $B$$$i$$j$)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-plus-ith*[*simp*]:($A + B$) $$ i = A $$ i + B $$ i
unfolding *sq-mtx-plus-eq* **by** (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-uminus-eq*: − A = *to-mtx* (χ i j. − $A$$$i$$j$)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-minus-eq*: $A - B$ = *to-mtx* (χ i j. $A$$$i$$j$ − $B$$$i$$j$)

by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-minus-ith*[*simp*]:$(A - B)$ \$\$ $i = A$ \$\$ $i - B$ \$\$ i
unfolding *sq-mtx-minus-eq* **by** (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-times-eq*: $A * B =$ *to-mtx* (χ i j. *sum* (λk. $A\$\$i\$k * B\$\$k\$j$) *UNIV*)
by *transfer* (*simp add*: *matrix-matrix-mult-def*)

lemma *sq-mtx-plus-diag-diag*[*simp*]: *sq-mtx-diag* f + *sq-mtx-diag* g = (diag i. f i + g i)
by (*subst sq-mtx-eq-iff*) (*simp add*: *axis-def*)

lemma *sq-mtx-minus-diag-diag*[*simp*]: *sq-mtx-diag* f − *sq-mtx-diag* g = (diag i. f i − g i)
by (*subst sq-mtx-eq-iff*) (*simp add*: *axis-def*)

lemma *sum-sq-mtx-diag*[*simp*]: ($\sum n<m$. *sq-mtx-diag* (g n)) = (diag i. $\sum n<m$. (g n i)) **for** m::*nat*
by (*induct m*, *simp*, *subst sq-mtx-eq-iff*, *simp-all*)

lemma *sq-mtx-mult-diag-diag*[*simp*]: *sq-mtx-diag* f ∗ *sq-mtx-diag* g = (diag i. f i ∗ g i)
by (*simp add*: *matrix-mul-diag-diag sq-mtx-diag.abs-eq times-sq-mtx.abs-eq*)

lemma *sq-mtx-mult-diagl*: (diag i. f i) ∗ A = *to-mtx* (χ i j. f i ∗ A \$\$ i \$ j)
by *transfer* (*simp add*: *matrix-mul-diag-matl*)

lemma *sq-mtx-mult-diagr*: A ∗ (diag i. f i) = *to-mtx* (χ i j. A \$\$ i \$ j ∗ f j)
by *transfer* (*simp add*: *matrix-matrix-mul-diag-matr*)

lemma *mtx-vec-mult-0l*[*simp*]: $0 *_V x = 0$
by (*simp add*: *sq-mtx-vec-mult.abs-eq zero-sq-mtx-def*)

lemma *mtx-vec-mult-0r*[*simp*]: $A *_V 0 = 0$
by (*transfer*, *simp*)

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

lemma *mtx-vec-mult-add-rdistl*: $A *_V (x + y) = A *_V x + A *_V y$
unfolding *plus-sq-mtx-def* **apply** *transfer*
by (*simp add*: *matrix-vector-right-distrib*)

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

lemma *mtx-vec-mult-minus-ldistrib*: $A *_V (x - y) = A *_V x - A *_V y$

by (*metis* (*no-types*, *lifting*) *add-diff-cancel diff-add-cancel matrix-vector-right-distrib sq-mtx-vec-mult.rep-eq*)

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

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

0.6.3 Real normed vector space of square matrices

instantiation *sq-mtx* :: (*finite*) *real-normed-vector*
begin

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

lift-definition *scaleR-sq-mtx* :: *real* $\Rightarrow$ $'a$ *sq-mtx* $\Rightarrow$ $'a$ *sq-mtx* **is** *scaleR* **.**

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

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

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

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

instance apply *intro-classes*
unfolding *sgn-sq-mtx-def open-sq-mtx-def dist-sq-mtx-def uniformity-sq-mtx-def*
prefer *10* **apply**(*transfer*, *simp add*: *norm-sq-mtx-def op-norm-triangle*)
prefer *9* **apply**(*simp-all add*: *norm-sq-mtx-def zero-sq-mtx-def op-norm-eq-0*)
by(*transfer*, *simp add*: *norm-sq-mtx-def op-norm-scaleR algebra-simps*)+

end

lemma *sq-mtx-scaleR-eq*: $c *_R A = to\text{-}mtx\ (\chi\ i\ j.\ c *_R A\ \$\$\ i\ \$\ j)$
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *scaleR-to-mtx-ith*[*simp*]: $c *_R (to\text{-}mtx\ A)\ \$\$\ i1\ \$\ i2 = c * A\ \$\ i1\ \$\ i2$
by *transfer* (*simp add*: *scaleR-vec-def*)

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

lemma *scaleR-sq-mtx-diag*: $c *_R$ *sq-mtx-diag* $f = (\mathrm{diag}\ i.\ c * f\ i)$
by (*subst sq-mtx-eq-iff*, *simp add*: *axis-def*)

lemma *scaleR-mtx-vec-assoc*: $(c *_R A) *_V x = c *_R (A *_V x)$
unfolding *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply** *simp*
by (*simp add*: *scaleR-matrix-vector-assoc*)

lemma *mtx-vec-scaleR-commute*: $A *_V (c *_R x) = c *_R (A *_V x)$
unfolding *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply**(*simp*, *transfer*)
by (*simp add*: *vector-scaleR-commute*)

lemma *mtx-times-scaleR-commute*: $A * (c *_R B) = c *_R (A * B)$ **for** *A*::(*'n*::*finite*) *sq-mtx*
unfolding *sq-mtx-scaleR-eq sq-mtx-times-eq* **apply**(*simp add*: *to-mtx-inject*)
apply(*simp add*: *vec-eq-iff fun-eq-iff*)
by (*simp add*: *semiring-normalization-rules*(*19*) *vector-space-over-itself.scale-sum-right*)

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

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

lemma *bounded-bilinear-sq-mtx-vec-mult*: *bounded-bilinear* $(\lambda A\ s.\ A *_V s)$
apply (*rule bounded-bilinear.intro*, *simp-all add*: *mtx-vec-mult-add-rdistr*
mtx-vec-mult-add-rdistl scaleR-mtx-vec-assoc mtx-vec-scaleR-commute)
by (*rule-tac x=1* **in** *exI*, *auto intro*!: *norm-vec-mult-le*)

lemma *norm-sq-mtx-def2*: $\|A\| = \textit{Sup}\ \{\|A *_V x\| \,|x.\ \|x\| = 1\}$
unfolding *norm-sq-mtx-def op-norm-def sq-mtx-vec-mult-def* **by** *simp*

lemma *norm-sq-mtx-def3*: $\|A\| = (\textit{SUP } x.\ (\|A *_V x\|) \,/\, (\|x\|))$
unfolding *norm-sq-mtx-def onorm-def sq-mtx-vec-mult-def* **by** *simp*

lemma *norm-sq-mtx-diag*: $\|\textit{sq-mtx-diag } f\| = \textit{Max}\ \{|f\ i| \,|i.\ i \in \textit{UNIV}\}$
unfolding *norm-sq-mtx-def* **apply** *transfer*
by (*rule op-norm-diag-mat-eq*)

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

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

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

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

0.6.4 Real normed algebra of square matrices

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

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

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

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

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

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

end

lemma *sq-mtx-one-ith-simps*[*simp*]: 1 \$\$ i \$ $i = 1$ $i \neq j \Longrightarrow 1$ \$\$ i \$ $j = 0$
unfolding *one-sq-mtx-def mat-def* **by** *simp-all*

lemma *of-nat-eq-sq-mtx-diag*[*simp*]: *of-nat* m = (diag i. m)
by (*induct m*) (*simp*, *subst sq-mtx-eq-iff*, *simp add*: *axis-def*)+

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

lemma *sq-mtx-diag-one*[*simp*]: (diag i. 1) = 1
by (*subst sq-mtx-eq-iff*, *simp add*: *one-sq-mtx-def mat-def axis-def*)

abbreviation *mtx-invertible* $A \equiv$ *invertible* (*to-vec* A)

lemma *mtx-invertible-def*: *mtx-invertible* $A \longleftrightarrow (\exists A'.\ A' * A = 1 \wedge A * A' = 1)$
apply (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def invertible-def*, *clarsimp*, *safe*)
apply(*rule-tac x=to-mtx A'* **in** *exI*, *simp*)
by (*rule-tac x=to-vec A'* **in** *exI*, *simp add*: *to-mtx-inject*)

lemma *mtx-invertibleI*:
assumes $A * B = 1$ **and** $B * A = 1$
shows *mtx-invertible* A
using *assms* **unfolding** *mtx-invertible-def* **by** *auto*

lemma *mtx-invertibleD*[*simp*]:
assumes *mtx-invertible A*
shows $A^{-1} * A = 1$ **and** $A * A^{-1} = 1$
apply (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def*)
using *assms* **by** *simp-all*

lemma *mtx-invertible-inv*[*simp*]: *mtx-invertible A* $\Longrightarrow$ *mtx-invertible* (A^{-1})
using *mtx-invertibleD mtx-invertibleI* **by** *blast*

lemma *mtx-invertible-one*[*simp*]: *mtx-invertible 1*
by (*simp add*: *one-sq-mtx.rep-eq*)

lemma *sq-mtx-inv-unique*:
assumes $A * B = 1$ **and** $B * A = 1$
shows $A^{-1} = B$
by (*metis* (*no-types*, *lifting*) *assms mtx-invertibleD*(*2*)
mtx-invertibleI mult.assoc sq-mtx-one-idty(*1*))

lemma *sq-mtx-inv-idempotent*[*simp*]: *mtx-invertible A* $\Longrightarrow A^{-1-1} = A$
using *mtx-invertibleD sq-mtx-inv-unique* **by** *blast*

lemma *sq-mtx-inv-mult*:
assumes *mtx-invertible A* **and** *mtx-invertible B*
shows $(A * B)^{-1} = B^{-1} * A^{-1}$
by (*simp add*: *assms matrix-inv-matrix-mul sq-mtx-inv-def times-sq-mtx-def*)

lemma *sq-mtx-inv-one*[*simp*]: $1^{-1} = 1$
by (*simp add*: *sq-mtx-inv-unique*)

definition *similar-sq-mtx* :: (*'n::finite*) *sq-mtx* $\Rightarrow$ *'n sq-mtx* $\Rightarrow$ *bool* (**infixr** $\sim$ *25*)
where $(A \sim B) \longleftrightarrow (\exists\ P.$ *mtx-invertible* $P \wedge A = P^{-1} * B * P)$

lemma *similar-sq-mtx-matrix*: $(A \sim B) =$ *similar-matrix* (*to-vec A*) (*to-vec B*)
apply(*unfold similar-matrix-def similar-sq-mtx-def*)
by (*smt UNIV-I to-mtx-inverse sq-mtx-inv.abs-eq times-sq-mtx.abs-eq to-vec-inverse*)

lemma *similar-sq-mtx-refl*[*simp*]: $A \sim A$
by (*unfold similar-sq-mtx-def*, *rule-tac x=1* **in** *exI*, *simp*)

lemma *similar-sq-mtx-simm*: $A \sim B \Longrightarrow B \sim A$
apply(*unfold similar-sq-mtx-def*, *clarsimp*)
apply(*rule-tac* $x{=}P^{-1}$ **in** *exI*, *simp add*: *mult.assoc*)
by (*metis mtx-invertibleD*(*2*) *mult.assoc mult.left-neutral*)

lemma *similar-sq-mtx-trans*: $A \sim B \Longrightarrow B \sim C \Longrightarrow A \sim C$
unfolding *similar-sq-mtx-matrix* **using** *similar-matrix-trans* **by** *blast*

lemma *power-sq-mtx-diag*: (*sq-mtx-diag f*) ^*n* = (diag *i*. *f i* ^*n*)

by (*induct n*, *simp-all*)

lemma *power-similiar-sq-mtx-diag-eq*:
assumes *mtx-invertible P*
and $A = P^{-1} * (sq\text{-}mtx\text{-}diag\ f) * P$
shows $A\,\hat{}\,n = P^{-1} * (\text{diag } i.\ f\ i\hat{}n) * P$
proof(*induct n*, *simp-all add*: *assms*)
fix *n*::*nat*
have $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (\text{diag } i.\ f\ i\ \hat{}\ n) * P) =$
$P^{-1} * sq\text{-}mtx\text{-}diag\ f * (\text{diag } i.\ f\ i\ \hat{}\ n) * P$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *mtx-invertibleD*(*2*) *mult.assoc mult.right-neutral*)
also have ... $= P^{-1} * (\text{diag } i.\ f\ i * f\ i\ \hat{}\ n) * P$
by (*simp add*: *mult.assoc*)
finally show $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (\text{diag } i.\ f\ i\ \hat{}\ n) * P) =$
$P^{-1} * (\text{diag } i.\ f\ i * f\ i\ \hat{}\ n) * P$ **.**
qed

lemma *power-similar-sq-mtx-diag*:
assumes $A \sim (sq\text{-}mtx\text{-}diag\ f)$
shows $A\,\hat{}\,n \sim (\text{diag } i.\ f\ i\hat{}n)$
using *assms power-similiar-sq-mtx-diag-eq*
unfolding *similar-sq-mtx-def* **by** *blast*

0.6.5 Banach space of square matrices

lemma *Cauchy-cols*:
fixes $X :: nat \Rightarrow ('a::finite)\ sq\text{-}mtx$
assumes *Cauchy X*
shows *Cauchy* ($\lambda n.$ col i $(X\ n)$)
proof(*unfold Cauchy-def dist-norm*, *clarsimp*)
fix ε::*real* **assume** $\varepsilon > 0$
then obtain M **where** *M-def*:$\forall m{\geq}M.\ \forall n{\geq}M.\ \|X\ m - X\ n\| < \varepsilon$
using ‹*Cauchy X*› **unfolding** *Cauchy-def* **by**(*simp add*: *dist-sq-mtx-def*) *metis*
{fix $m\ n$ **assume** $m \geq M$ **and** $n \geq M$
hence $\varepsilon > \|X\ m - X\ n\|$
using *M-def* **by** *blast*
moreover have $\|X\ m - X\ n\| \geq \|(X\ m - X\ n) *_V \text{e}\ i\|$
by(*rule le-mtx-norm*[*of* - $X\ m - X\ n$], *force*)
moreover have $\|(X\ m - X\ n) *_V \text{e}\ i\| = \|X\ m *_V \text{e}\ i - X\ n *_V \text{e}\ i\|$
by (*simp add*: *mtx-vec-mult-minus-rdistrib*)
moreover have ... $= \|\text{col } i\ (X\ m) - \text{col } i\ (X\ n)\|$
by (*simp add*: *mtx-vec-mult-minus-rdistrib mtx-vec-mult-canon*)
ultimately have $\|\text{col } i\ (X\ m) - \text{col } i\ (X\ n)\| < \varepsilon$
by *linarith***}**
thus $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ \|\text{col } i\ (X\ m) - \text{col } i\ (X\ n)\| < \varepsilon$
by *blast*
qed

lemma *col-convergence*:

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

instance *sq-mtx* :: (*finite*) *banach*
proof(*standard*)
fix X :: *nat* $\Rightarrow$ *'a sq-mtx*
assume *Cauchy X*
hence $\bigwedge i.$ *Cauchy* $(\lambda n.\ \mathrm{col}\ i\ (X\ n))$
using *Cauchy-cols* **by** *blast*
hence *obs*: $\forall i.\ \exists!\ L.\ (\lambda n.\ \mathrm{col}\ i\ (X\ n)) \longrightarrow L$
using *Cauchy-convergent convergent-def LIMSEQ-unique* **by** *fastforce*
define L **where** $L = (\chi\ i.\ lim\ (\lambda n.\ \mathrm{col}\ i\ (X\ n)))$
hence $\forall i.\ (\lambda n.\ \mathrm{col}\ i\ (X\ n)) \longrightarrow L\ \$\ i$
using *obs theI-unique*[*of* $\lambda L.\ (\lambda n.\ \mathrm{col}\ \text{-}\ (X\ n)) \longrightarrow L\ L\ \$\ \text{-}$] **by** (*simp add*: *lim-def*)
thus *convergent X*
using *col-convergence* **unfolding** *convergent-def* **by** *blast*
qed

lemma *exp-similiar-sq-mtx-diag-eq*:
assumes *mtx-invertible* P
and $A = P^{-1} * (\text{diag } i.\ f\ i) * P$
shows *exp* $A = P^{-1} *$ *exp* $(\text{diag } i.\ f\ i) * P$
proof(*unfold exp-def power-similiar-sq-mtx-diag-eq*[*OF assms*])
have $(\sum n.\ P^{-1} * (\text{diag } i.\ f\ i\ \hat{}\ n) * P\ /_R\ \mathit{fact}\ n) =$
$(\sum n.\ P^{-1} * ((\text{diag } i.\ f\ i\ \hat{}\ n)\ /_R\ \mathit{fact}\ n) * P)$
by *simp*
also have ... $= (\sum n.\ P^{-1} * ((\text{diag } i.\ f\ i\ \hat{}\ n)\ /_R\ \mathit{fact}\ n)) * P$
apply(*subst suminf-multr*[*OF bounded-linear.summable*[*OF bounded-linear-mult-right*]])
unfolding *power-sq-mtx-diag*[*symmetric*] **by** (*simp-all add*: *summable-exp-generic*)
also have ... $= P^{-1} * (\sum n.\ (\text{diag } i.\ f\ i\ \hat{}\ n)\ /_R\ \mathit{fact}\ n) * P$
apply(*subst suminf-mult*[*of* - P^{-1}])
unfolding *power-sq-mtx-diag*[*symmetric*]
by (*simp-all add*: *summable-exp-generic*)
finally show $(\sum n.\ P^{-1} * (\text{diag } i.\ f\ i\ \hat{}\ n) * P\ /_R\ \mathit{fact}\ n) =$
$P^{-1} * (\sum n.\ \textit{sq-mtx-diag}\ f\ \hat{}\ n\ /_R\ \mathit{fact}\ n) * P$
unfolding *power-sq-mtx-diag* **by** *simp*
qed

lemma *exp-similiar-sq-mtx-diag*:
assumes $A \sim$ *sq-mtx-diag f*
shows *exp* $A \sim$ *exp* (*sq-mtx-diag f*)
using *assms exp-similiar-sq-mtx-diag-eq*
unfolding *similar-sq-mtx-def* **by** *blast*

lemma *suminf-sq-mtx-diag*:
assumes $\forall i.\ (\lambda n.\ f\ n\ i)$ *sums* (*suminf* $(\lambda n.\ f\ n\ i)$)
shows $(\sum n.\ (\text{diag } i.\ f\ n\ i)) = (\text{diag } i.\ \sum n.\ f\ n\ i)$
proof(*rule suminfI*, *unfold sums-def LIMSEQ-iff*, *clarsimp simp*: *norm-sq-mtx-diag*)
let $?g = \lambda n\ i.\ |(\sum n{<}n.\ f\ n\ i) - (\sum n.\ f\ n\ i)|$
fix r::*real* **assume** $r > 0$
have $\forall i.\ \exists no.\ \forall n{\ge}no.\ ?g\ n\ i < r$
using *assms* ‹$r > 0$› **unfolding** *sums-def LIMSEQ-iff* **by** *clarsimp*
then obtain N **where** *key*: $\forall i.\ \forall n{\ge}N.\ ?g\ n\ i < r$
using *finite-nat-minimal-witness*[*of* $\lambda i\ n.\ ?g\ n\ i < r$] **by** *blast*
{fix n::*nat*
assume $n \ge N$
obtain i **where** *i-def*: $Max\ \{x.\ \exists i.\ x = ?g\ n\ i\} = ?g\ n\ i$
using *cMax-finite-ex*[*of* $\{x.\ \exists i.\ x = ?g\ n\ i\}$] **by** *auto*
hence $?g\ n\ i < r$
using *key* ‹$n \ge N$› **by** *blast*
hence $Max\ \{x.\ \exists i.\ x = ?g\ n\ i\} < r$
unfolding *i-def*[*symmetric*] **.}**
thus $\exists N.\ \forall n{\ge}N.\ Max\ \{x.\ \exists i.\ x = ?g\ n\ i\} < r$
by *blast*
qed

lemma *exp-sq-mtx-diag*: *exp* (*sq-mtx-diag f*) = (diag *i*. *exp* (*f i*))
apply(*unfold exp-def*, *simp add*: *power-sq-mtx-diag scaleR-sq-mtx-diag*)
apply(*rule suminf-sq-mtx-diag*)
using *exp-converges*[*of f* -]
unfolding *sums-def LIMSEQ-iff exp-def* **by** *force*

lemma *exp-scaleR-diagonal1*:
assumes *mtx-invertible P* **and** $A = P^{-1} * (\text{diag } i.\ f\ i) * P$
shows $exp\ (t *_R A) = P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$
proof−
have $exp\ (t *_R A) = exp\ (P^{-1} * (t *_R sq\text{-}mtx\text{-}diag\ f) * P)$
using *assms* **by** *simp*
also have ... $= P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$
by (*metis assms*(*1*) *exp-similiar-sq-mtx-diag-eq exp-sq-mtx-diag scaleR-sq-mtx-diag*)
finally show $exp\ (t *_R A) = P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$ **.**
qed

lemma *exp-scaleR-diagonal2*:
assumes *mtx-invertible P* **and** $A = P * (\text{diag } i.\ f\ i) * P^{-1}$
shows $exp\ (t *_R A) = P * (\text{diag } i.\ exp\ (t * f\ i)) * P^{-1}$
apply(*subst sq-mtx-inv-idempotent*[*OF assms*(*1*), *symmetric*])
apply(*rule exp-scaleR-diagonal1*)
by (*simp-all add*: *assms*)

0.6.6 Examples

definition *mtx A* = *to-mtx* (*vector* (*map vector A*))

lemma *vector-nth-eq*: (*vector A*) \$ *i* = *foldr* ($\lambda x\ f\ n.\ (f\ (n + 1))(n := x)$) *A* ($\lambda n$ *x*. *0*) *1 i*
unfolding *vector-def* **by** *simp*

lemma *mtx-ith-eq*[*simp*]: *mtx A* \$\$ *i* \$ *j* = *foldr* ($\lambda x\ f\ n.\ (f\ (n + 1))(n := x)$)
(*map* (λl. *vec-lambda* (*foldr* ($\lambda x\ f\ n.\ (f\ (n + 1))(n := x)$) *l* ($\lambda n\ x.\ 0$) *1*)) *A*)
($\lambda n\ x.\ 0$) *1 i* \$ *j*
unfolding *mtx-def vector-def* **by** (*simp add*: *vector-nth-eq*)

2x2 matrices

lemma *mtx2-eq-iff*: (*mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) :: *2 sq-mtx*) = *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) $\longleftrightarrow$ $a1 = a2 \wedge b1 = b2 \wedge c1 = c2 \wedge d1 = d2$
apply(*simp add*: *sq-mtx-eq-iff*, *safe*)
using *exhaust-2* **by** *force*+

lemma *mtx2-to-mtx*: *mtx*
([*a*, *b*] #
[*c*, *d*] # []) =

to-mtx (χ *i j*::*2*. *if i*=*1* $\wedge$ *j*=*1 then a*
else (*if i*=*1* $\wedge$ *j*=*2 then b*
else (*if i*=*2* $\wedge$ *j*=*1 then c*
else d)))
apply(*subst sq-mtx-eq-iff*)
using *exhaust-2* **by** *force*

abbreviation *diag2* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *2 sq-mtx*
where *diag2* ι_1 ι_2 $\equiv$ *mtx*
([ι_1, *0*] #
[*0*, ι_2] # [])

lemma *diag2-eq*: *diag2* (ι *1*) (ι *2*) = (diag *i*. ι *i*)
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-2* **by** (*force simp*: *axis-def*)

lemma *one-mtx2*: (*1*::*2 sq-mtx*) = *diag2 1 1*
apply(*subst sq-mtx-eq-iff*)
using *exhaust-2* **by** *force*

lemma *zero-mtx2*: (*0*::*2 sq-mtx*) = *diag2 0 0*
by (*simp add*: *sq-mtx-eq-iff*)

lemma *scaleR-mtx2*: *k* $*_R$ *mtx*
([*a*, *b*] #
[*c*, *d*] # []) = *mtx*
([*k*∗*a*, *k*∗*b*] #
[*k*∗*c*, *k*∗*d*] # [])
by (*simp add*: *sq-mtx-eq-iff*)

lemma *uminus-mtx2*: −*mtx*
([*a*, *b*] #
[*c*, *d*] # []) = (*mtx*
([−*a*, −*b*] #
[−*c*, −*d*] # [])::*2 sq-mtx*)
by (*simp add*: *sq-mtx-uminus-eq sq-mtx-eq-iff*)

lemma *plus-mtx2*: *mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) + *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) = ((*mtx*
([*a1*+*a2*, *b1*+*b2*] #
[*c1*+*c2*, *d1*+*d2*] # []))::*2 sq-mtx*)
by (*simp add*: *sq-mtx-eq-iff*)

lemma *minus-mtx2*: *mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) − *mtx*

([$a2$, $b2$] #
[$c2$, $d2$] # []) = ((*mtx*
([$a1-a2$, $b1-b2$] #
[$c1-c2$, $d1-d2$] # []))::*2 sq-mtx*)
by (*simp add*: *sq-mtx-eq-iff*)

lemma *times-mtx2*: *mtx*
([$a1$, $b1$] #
[$c1$, $d1$] # []) ∗ *mtx*
([$a2$, $b2$] #
[$c2$, $d2$] # []) = ((*mtx*
([$a1*a2+b1*c2$, $a1*b2+b1*d2$] #
[$c1*a2+d1*c2$, $c1*b2+d1*d2$] # []))::*2 sq-mtx*)
unfolding *sq-mtx-times-eq UNIV-2*
by (*simp add*: *sq-mtx-eq-iff*)

3x3 matrices

lemma *mtx3-to-mtx*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) =
to-mtx (χ *i j*::*3*. *if* $i=1 \wedge j=1$ *then* a_{11}
else (*if* $i=1 \wedge j=2$ *then* a_{12}
else (*if* $i=1 \wedge j=3$ *then* a_{13}
else (*if* $i=2 \wedge j=1$ *then* a_{21}
else (*if* $i=2 \wedge j=2$ *then* a_{22}
else (*if* $i=2 \wedge j=3$ *then* a_{23}
else (*if* $i=3 \wedge j=1$ *then* a_{31}
else (*if* $i=3 \wedge j=2$ *then* a_{32}
else a_{33}))))))))
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-3* **by** *force*

abbreviation *diag3* :: *real* ⇒ *real* ⇒ *real* ⇒ *3 sq-mtx*
where *diag3* ι_1 ι_2 ι_3 ≡ *mtx*
([ι_1, *0*, *0*] #
[*0*, ι_2, *0*] #
[*0*, *0*, ι_3] # [])

lemma *diag3-eq*: *diag3* (ι *1*) (ι *2*) (ι *3*) = (diag *i*. ι *i*)
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-3* **by** (*force simp*: *axis-def*)

lemma *one-mtx3*: (*1*::*3 sq-mtx*) = *diag3 1 1 1*
apply(*subst sq-mtx-eq-iff*)
using *exhaust-3* **by** *force*

lemma *zero-mtx3*: (*0*::*3 sq-mtx*) = *diag3 0 0 0*

by (*simp add*: *sq-mtx-eq-iff*)

lemma *scaleR-mtx3*: $k *_R$ *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) = *mtx*
([$k*a_{11}$, $k*a_{12}$, $k*a_{13}$] #
[$k*a_{21}$, $k*a_{22}$, $k*a_{23}$] #
[$k*a_{31}$, $k*a_{32}$, $k*a_{33}$] # [])
by (*simp add*: *sq-mtx-eq-iff*)

lemma *plus-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) + *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([$a_{11}+b_{11}$, $a_{12}+b_{12}$, $a_{13}+b_{13}$] #
[$a_{21}+b_{21}$, $a_{22}+b_{22}$, $a_{23}+b_{23}$] #
[$a_{31}+b_{31}$, $a_{32}+b_{32}$, $a_{33}+b_{33}$] # [])::*3 sq-mtx*)
by (*subst sq-mtx-eq-iff*) *simp*

lemma *minus-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) − *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([$a_{11}-b_{11}$, $a_{12}-b_{12}$, $a_{13}-b_{13}$] #
[$a_{21}-b_{21}$, $a_{22}-b_{22}$, $a_{23}-b_{23}$] #
[$a_{31}-b_{31}$, $a_{32}-b_{32}$, $a_{33}-b_{33}$] # [])::*3 sq-mtx*)
by (*simp add*: *sq-mtx-eq-iff*)

lemma *times-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) * *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([$a_{11}*b_{11}+a_{12}*b_{21}+a_{13}*b_{31}$, $a_{11}*b_{12}+a_{12}*b_{22}+a_{13}*b_{32}$, $a_{11}*b_{13}+a_{12}*b_{23}+a_{13}*b_{33}$]
#
[$a_{21}*b_{11}+a_{22}*b_{21}+a_{23}*b_{31}$, $a_{21}*b_{12}+a_{22}*b_{22}+a_{23}*b_{32}$, $a_{21}*b_{13}+a_{22}*b_{23}+a_{23}*b_{33}$]
#
[$a_{31}*b_{11}+a_{32}*b_{21}+a_{33}*b_{31}$, $a_{31}*b_{12}+a_{32}*b_{22}+a_{33}*b_{32}$, $a_{31}*b_{13}+a_{32}*b_{23}+a_{33}*b_{33}$]
[])::*3 sq-mtx*)
unfolding *sq-mtx-times-eq*

unfolding *UNIV-3* **by** (*simp add*: *sq-mtx-eq-iff*)

end

0.7 Affine systems of ODEs

Affine systems of ordinary differential equations (ODEs) are those whose vector fields are linear operators. Broadly speaking, if there are functions A and B such that the system of ODEs $X'\,t = f\,(X\,t)$ turns into $X'\,t = (A\,t)\cdot(X\,t) + (B\,t)$, then it is affine. The end goal of this section is to prove that every affine system of ODEs has a unique solution, and to obtain a characterization of said solution.

theory *mtx-flows*
 imports *mtx-sq hs-prelims-dyn-sys*

begin

0.7.1 Existence and uniqueness for affine systems

definition *matrix-continuous-on* :: *real set* $\Rightarrow$ (*real* $\Rightarrow$ ($'a$::*real-normed-algebra-1*)$\hat{\ }'n\hat{\ }'m$) $\Rightarrow$ *bool*
 where *matrix-continuous-on T A* = ($\forall\, t \in T.\ \forall \varepsilon > 0.\ \exists\ \delta > 0.\ \forall \tau{\in}T.\ |\tau - t| < \delta \longrightarrow \|A\ \tau - A\ t\|_{op} \leq \varepsilon$)

lemma *continuous-on-matrix-vector-multl*:
 assumes *matrix-continuous-on T A*
 shows *continuous-on T* ($\lambda t.\ A\ t\ {*}v\ s$)
proof(*rule continuous-onI*, *simp add*: *dist-norm*)
 fix *e t*::*real* **assume** $0 < e$ **and** $t \in T$
 let $?\varepsilon = e/(\|(\mathit{if}\ s = 0\ \mathit{then}\ 1\ \mathit{else}\ s)\|)$
 have $?\varepsilon > 0$
  using ‹$0 < e$› **by** *simp*
 then obtain δ **where** *dHyp*: $\delta > 0 \wedge (\forall \tau{\in}T.\ |\tau - t| < \delta \longrightarrow \|A\ \tau - A\ t\|_{op} \leq\ ?\varepsilon)$
  using *assms* ‹$t \in T$› **unfolding** *dist-norm matrix-continuous-on-def* **by** *fastforce*
 {**fix** τ **assume** $\tau \in T$ **and** $|\tau - t| < \delta$
  have *obs*: $?\varepsilon * (\|s\|) = (\mathit{if}\ s = 0\ \mathit{then}\ 0\ \mathit{else}\ e)$
   by *auto*
  have $\|A\ \tau\ {*}v\ s - A\ t\ {*}v\ s\| = \|(A\ \tau - A\ t)\ {*}v\ s\|$
   by (*simp add*: *matrix-vector-mult-diff-rdistrib*)
  also have ... $\leq (\|A\ \tau - A\ t\|_{op}) * (\|s\|)$
   using *norm-matrix-le-mult-op-norm* **by** *blast*
  also have ... $\leq\ ?\varepsilon * (\|s\|)$
   using *dHyp* ‹$\tau \in T$› ‹$|\tau - t| < \delta$› *mult-right-mono norm-ge-zero* **by** *blast*
  finally have $\|A\ \tau\ {*}v\ s - A\ t\ {*}v\ s\| \leq e$
   by (*subst* (*asm*) *obs*) (*metis* (*mono-tags*, *hide-lams*) ‹$0 < e$› *less-eq-real-def order-trans*)}
 thus $\exists\, d{>}0.\ \forall \tau{\in}T.\ |\tau - t| < d \longrightarrow \|A\ \tau\ {*}v\ s - A\ t\ {*}v\ s\| \leq e$

```
    using dHyp by blast
qed

lemma lipschitz-cond-affine:
  fixes A :: real ⇒ 'a::real-normed-algebra-1^'n^'m and T::real set
  defines L ≡ Sup {‖A t‖_op |t. t ∈ T}
  assumes t ∈ T and bdd-above {‖A t‖_op |t. t ∈ T}
  shows ‖A t *v x − A t *v y‖ ≤ L * (‖x − y‖)
proof−
  have obs: ‖A t‖_op ≤ Sup {‖A t‖_op |t. t ∈ T}
    apply(rule cSup-upper)
    using continuous-on-subset assms by (auto simp: dist-norm)
  have ‖A t *v x − A t *v y‖ = ‖A t *v (x − y)‖
    by (simp add: matrix-vector-mult-diff-distrib)
  also have ... ≤ (‖A t‖_op) * (‖x − y‖)
    using norm-matrix-le-mult-op-norm by blast
  also have ... ≤ Sup {‖A t‖_op |t. t ∈ T} * (‖x − y‖)
    using obs mult-right-mono norm-ge-zero by blast
  finally show ‖A t *v x − A t *v y‖ ≤ L * (‖x − y‖)
    unfolding assms .
qed

lemma local-lipschitz-affine:
  fixes A :: real ⇒ 'a::real-normed-algebra-1^'n^'m
  assumes open T and open S
    and Ahyp: ⋀τ ε. ε > 0 ⟹ τ ∈ T ⟹ cball τ ε ⊆ T ⟹ bdd-above {‖A t‖_op
|t. t ∈ cball τ ε}
  shows local-lipschitz T S (λt s. A t *v s + B t)
proof(unfold local-lipschitz-def lipschitz-on-def, clarsimp)
  fix s t assume s ∈ S and t ∈ T
  then obtain e1 e2 where cball t e1 ⊆ T and cball s e2 ⊆ S and min e1 e2 >
0
    using open-cballE[OF - ‹open T›] open-cballE[OF - ‹open S›] by force
  hence obs: cball t (min e1 e2) ⊆ T
    by auto
  let ?L = Sup {‖A τ‖_op |τ. τ ∈ cball t (min e1 e2)}
  have ‖A t‖_op ∈ {‖A τ‖_op |τ. τ ∈ cball t (min e1 e2)}
    using ‹min e1 e2 > 0› by auto
  moreover have bdd: bdd-above {‖A τ‖_op |τ. τ ∈ cball t (min e1 e2)}
    by (rule Ahyp, simp only: ‹min e1 e2 > 0›, simp-all add: ‹t ∈ T› obs)
  moreover have Sup {‖A τ‖_op |τ. τ ∈ cball t (min e1 e2)} ≥ 0
    apply(rule order.trans[OF op-norm-ge-0[of A t]])
    by (rule cSup-upper[OF calculation])
  moreover have ∀x∈cball s (min e1 e2) ∩ S. ∀y∈cball s (min e1 e2) ∩ S.
    ∀τ∈cball t (min e1 e2) ∩ T. dist (A τ *v x) (A τ *v y) ≤ ?L * dist x y
    apply(clarify, simp only: dist-norm, rule lipschitz-cond-affine)
    using ‹min e1 e2 > 0› bdd by auto
  ultimately show ∃e>0. ∃L. ∀t∈cball t e ∩ T. 0 ≤ L ∧
   (∀x∈cball s e ∩ S. ∀y∈cball s e ∩ S. dist (A t *v x) (A t *v y) ≤ L * dist x y)
```

using ‹*min e1 e2 > 0*› **by** *blast*
qed

lemma *picard-lindeloef-affine*:
fixes A :: *real* ⇒ *'a*::{*banach*,*real-normed-algebra-1*,*heine-borel*}ˆ*'n*ˆ*'n*
assumes *Ahyp*: *matrix-continuous-on T A*
and $\bigwedge\tau\ \varepsilon.\ \tau \in T \Longrightarrow \varepsilon > 0 \Longrightarrow$ *bdd-above* $\{\|A\ t\|_{op}\ |t.\ dist\ \tau\ t \leq \varepsilon\}$
and *Bhyp*: *continuous-on T B* **and** *open S*
and $t_0 \in T$ **and** *Thyp*: *open T is-interval T*
shows *picard-lindeloef* (λ *t s*. *A t* ∗*v s* + *B t*) *T S* t_0
apply(*unfold-locales*, *simp-all add*: *assms*, *clarsimp*)
apply(*rule continuous-on-add*[*OF continuous-on-matrix-vector-multl*[*OF Ahyp*] *Bhyp*])
by (*rule local-lipschitz-affine*) (*simp-all add*: *assms*)

lemma *picard-lindeloef-autonomous-affine*:
fixes A :: *'a*::{*banach*,*real-normed-field*,*heine-borel*}ˆ*'n*ˆ*'n*
shows *picard-lindeloef* (λ *t s*. *A* ∗*v s* + *B*) *UNIV UNIV* t_0
using *picard-lindeloef-affine*[*of* - λ*t*. *A* λ*t*. *B*]
unfolding *matrix-continuous-on-def* **by** (*simp only*: *diff-self op-norm0*, *auto*)

lemma *picard-lindeloef-autonomous-linear*:
fixes A :: *'a*::{*banach*,*real-normed-field*,*heine-borel*}ˆ*'n*ˆ*'n*
shows *picard-lindeloef* (λ *t*. (∗*v*) *A*) *UNIV UNIV* t_0
using *picard-lindeloef-autonomous-affine*[*of A 0*] **by** *force*

lemmas *unique-sol-autonomous-affine* = *picard-lindeloef*.*unique-solution*[*OF picard-lindeloef-autonomous-affine - - funcset-UNIV UNIV-I - - funcset-UNIV UNIV-I*]

lemmas *unique-sol-autonomous-linear* = *picard-lindeloef*.*unique-solution*[*OF picard-lindeloef-autonomous-linear - - funcset-UNIV UNIV-I - - funcset-UNIV UNIV-I*]

0.7.2 Flow for affine systems

Derivative rules for square matrices

lemma *has-derivative-exp-scaleRl*[*derivative-intros*]:
fixes *f*::*real* ⇒ *real*
assumes *D f* ↦ *f'* *at t within T*
shows *D* (λ*t*. *exp* (*f t* $*_R$ *A*)) ↦ (λ*h*. *f' h* $*_R$ (*exp* (*f t* $*_R$ *A*) ∗ *A*)) *at t within T*
proof −
have *bounded-linear f'*
using *assms* **by** *auto*
then obtain *m* **where** *obs*: *f'* = (λ*h*. *h* ∗ *m*)
using *real-bounded-linear* **by** *blast*
thus *?thesis*
using *vector-diff-chain-within*[*OF* - *exp-scaleR-has-vector-derivative-right*]

assms obs **by** (*auto simp*: *has-vector-derivative-def comp-def*)
qed

lemma *has-vderiv-on-exp-scaleRl*:
assumes $D\, f = f'$ *on* T
shows $D\ (\lambda x.\ exp\ (f\, x *_R A)) = (\lambda x.\ f'\ x *_R exp\ (f\, x *_R A) * A)$ *on* T
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
by (*rule has-derivative-exp-scaleRl*, *auto simp*: *fun-eq-iff*)

lemma *vderiv-on-exp-scaleRlI*[*poly-derivatives*]:
assumes $D\, f = f'$ *on* T **and** $g' = (\lambda x.\ f'\ x *_R exp\ (f\, x *_R A) * A)$
shows $D\ (\lambda x.\ exp\ (f\, x *_R A)) = g'$ *on* T
using *has-vderiv-on-exp-scaleRl assms* **by** *simp*

lemma *has-derivative-mtx-ith*[*derivative-intros*]:
fixes *t*::*real* **and** *T* :: *real set*
defines $t_0 \equiv$ *netlimit* (*at t within T*)
assumes $D\ A \mapsto (\lambda h.\ h *_R A'\ t)$ *at t within T*
shows $D\ (\lambda t.\ A\ t\ \$\$\ i) \mapsto (\lambda h.\ h *_R A'\ t\ \$\$\ i)$ *at t within T*
using *assms* **unfolding** *has-derivative-def* **apply** *safe*
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
apply(*rule-tac* $F{=}\lambda\tau.\ (A\ \tau - A\ t_0 - (\tau - t_0) *_R A'\ t)\ /_R\ (\|\tau - t_0\|)$ **in** *tendsto-zero-norm-bound*)
by (*clarsimp*, *rule mult-left-mono*, *metis* (*no-types*, *lifting*) *norm-column-le-norm*

sq-mtx-minus-ith sq-mtx-scaleR-ith) *simp-all*

lemmas *has-derivative-mtx-vec-mult*[*derivative-intros*] =
bounded-bilinear.FDERIV[*OF bounded-bilinear-sq-mtx-vec-mult*]

lemma *vderiv-mtx-vec-mult-intro*[*poly-derivatives*]:
assumes $D\ u = u'$ *on* T **and** $D\ A = A'$ *on* T
and $g = (\lambda t.\ A\ t *_V u'\ t + A'\ t *_V u\ t\ )$
shows $D\ (\lambda t.\ A\ t *_V u\ t) = g$ *on* T
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
apply(*erule-tac x*=*x* **in** *ballE*, *simp-all*)+
apply(*rule derivative-eq-intros*(*144*))
by (*auto simp*: *fun-eq-iff mtx-vec-scaleR-commute pth-6 scaleR-mtx-vec-assoc*)

lemmas *has-vderiv-on-ivl-integral* = *ivl-integral-has-vderiv-on*[*OF vderiv-on-continuous-on*]

declare *has-vderiv-on-ivl-integral* [*poly-derivatives*]

lemma *has-derivative-mtx-vec-multl*[*derivative-intros*]:
assumes $\bigwedge i\ j.\ D\ (\lambda t.\ (A\ t)\ \$\$\ i\ \$\ j) \mapsto (\lambda\tau.\ \tau *_R (A'\ t)\ \$\$\ i\ \$\ j)$ (*at t within T*)
shows $D\ (\lambda t.\ A\ t *_V x) \mapsto (\lambda\tau.\ \tau *_R (A'\ t) *_V x)$ *at t within T*
unfolding *sq-mtx-vec-mult-sum-cols*
apply(*rule-tac* $f'1{=}\lambda i\ \tau.\ \tau *_R (x\ \$\ i *_R \mathrm{col}\ i\ (A'\ t))$ **in** *derivative-eq-intros*(*10*))

apply(*simp-all add*: *scaleR-right.sum*)
apply(*rule-tac g'1*=$\lambda\tau$. $\tau *_R$ col *i* (A' *t*) **in** *derivative-eq-intros*(*4*), *simp-all add*: *mult.commute*)
using *assms* **unfolding** *sq-mtx-col-def column-def* **apply**(*transfer*, *simp*)
apply(*rule has-derivative-vec-lambda*)
by (*simp add*: *scaleR-vec-def*)

lemma *continuous-on-mtx-vec-multr*: *continuous-on S* $((*_V)\ A)$
by *transfer* (*simp add*: *matrix-vector-mult-linear-continuous-on*)

— Automatically generated derivative rules from this subsubsection

thm *derivative-eq-intros*(*142*,*143*,*144*,*145*)

Existence and uniqueness with square matrices

Finally, we can use the *exp* operation to characterize the general solutions for affine systems of ODEs. We show that they satisfy the *local-flow* locale.

lemma *continuous-on-sq-mtx-vec-multl*:
fixes *A* :: *real* ⇒ (*'n*::*finite*) *sq-mtx*
assumes *continuous-on T A*
shows *continuous-on T* (λt. $A\ t *_V s$)
proof−
have *matrix-continuous-on T* (λt. *to-vec* (*A t*))
using *assms* **by** (*force simp*: *continuous-on-iff dist-norm norm-sq-mtx-def matrix-continuous-on-def*)
hence *continuous-on T* (λt. *to-vec* (*A t*) *$*v$ *s*)
by (*rule continuous-on-matrix-vector-multl*)
thus *?thesis*
by *transfer*
qed

lemmas *continuous-on-affine* = *continuous-on-add*[*OF continuous-on-sq-mtx-vec-multl*]

lemma *local-lipschitz-sq-mtx-affine*:
fixes *A* :: *real* ⇒ (*'n*::*finite*) *sq-mtx*
assumes *continuous-on T A open T open S*
shows *local-lipschitz T S* ($\lambda t\ s$. $A\ t *_V s + B\ t$)
proof−
have *obs*: $\bigwedge\tau\ \varepsilon.\ 0 < \varepsilon \Longrightarrow\ \tau \in T \Longrightarrow$ *cball* $\tau\ \varepsilon \subseteq T \Longrightarrow$ *bdd-above* $\{\|A\ t\|\ |t.\ t \in$ *cball* $\tau\ \varepsilon\}$
by (*rule bdd-above-norm-cont-comp*, *rule continuous-on-subset*[*OF assms*(*1*)], *simp-all*)
hence $\bigwedge\tau\ \varepsilon.\ 0 < \varepsilon \Longrightarrow \tau \in T \Longrightarrow$ *cball* $\tau\ \varepsilon \subseteq T \Longrightarrow$ *bdd-above* $\{\|$*to-vec* (*A t*)$\|_{op}\ |t.\ t \in$ *cball* $\tau\ \varepsilon\}$
by (*simp add*: *norm-sq-mtx-def*)
hence *local-lipschitz T S* ($\lambda t\ s$. *to-vec* (*A t*) *$*v$ *s* + *B t*)
using *local-lipschitz-affine*[*OF assms*(*2*,*3*), *of* λt. *to-vec* (*A t*)] **by** *force*
thus *?thesis*

by *transfer*
qed

lemma *picard-lindeloef-sq-mtx-affine*:
assumes *continuous-on* T A **and** *continuous-on* T B
and $t_0 \in T$ *is-interval* T *open* T **and** *open* S
shows *picard-lindeloef* $(\lambda t\ s.\ A\ t *_V s + B\ t)\ T\ S\ t_0$
apply(*unfold-locales*, *simp-all add*: *assms*, *clarsimp*)
using *continuous-on-affine assms* **apply** *blast*
by (*rule local-lipschitz-sq-mtx-affine*, *simp-all add*: *assms*)

lemmas *sq-mtx-unique-sol-autonomous-affine* = *picard-lindeloef.unique-solution*[*OF*

picard-lindeloef-sq-mtx-affine[*OF*
continuous-on-const
continuous-on-const
UNIV-I is-interval-univ
open-UNIV open-UNIV]
- - *funcset-UNIV UNIV-I* - - *funcset-UNIV UNIV-I*]

lemma *has-vderiv-on-sq-mtx-linear*:
$D\ (\lambda t.\ exp\ ((t - t_0) *_R A) *_V s) = (\lambda t.\ A *_V (exp\ ((t - t_0) *_R A) *_V s))$ *on* $\{t_0{-}{-}t\}$
by (*rule poly-derivatives*)+ (*auto simp*: *exp-times-scaleR-commute sq-mtx-times-vec-assoc*)

lemma *has-vderiv-on-sq-mtx-affine*:
fixes t_0::*real* **and** A :: (*'a*::*finite*) *sq-mtx*
defines *lSol* $c\ t \equiv exp\ ((c * (t - t_0)) *_R A)$
shows $D\ (\lambda t.\ lSol\ 1\ t *_V s + lSol\ 1\ t *_V (\int_{t0}{}^{t} (lSol\ (-1)\ \tau *_V B)\ \partial\tau)) =$
$(\lambda t.\ A *_V (lSol\ 1\ t *_V s + lSol\ 1\ t *_V (\int_{t0}{}^{t} (lSol\ (-1)\ \tau *_V B)\ \partial\tau)) + B)$ *on* $\{t_0{-}{-}t\}$
unfolding *assms* **apply**(*simp only*: *mult.left-neutral mult-minus1*)
apply(*rule poly-derivatives*, (*force*)?, (*force*)?, (*force*)?, (*force*)?)+
by (*simp add*: *mtx-vec-mult-add-rdistl sq-mtx-times-vec-assoc*[*symmetric*]
exp-minus-inverse exp-times-scaleR-commute mult-exp-exp scale-left-distrib[*symmetric*])

lemma *autonomous-linear-sol-is-exp*:
assumes $D\ X = (\lambda t.\ A *_V X\ t)$ *on* $\{t_0{-}{-}t\}$ **and** $X\ t_0 = s$
shows $X\ t = exp\ ((t - t_0) *_R A) *_V s$
apply(*rule sq-mtx-unique-sol-autonomous-affine*[*of* X A 0, *OF* - ‹$X\ t_0 = s$›])
using *assms has-vderiv-on-sq-mtx-linear* **by** *force*+

lemma *autonomous-affine-sol-is-exp-plus-int*:
assumes $D\ X = (\lambda t.\ A *_V X\ t + B)$ *on* $\{t_0{-}{-}t\}$ **and** $X\ t_0 = s$
shows $X\ t = exp\ ((t - t_0) *_R A) *_V s + exp\ ((t - t_0) *_R A) *_V (\int_{t0}{}^{t}(exp\ (-(\tau - t_0) *_R A) *_V B)\partial\tau)$
apply(*rule sq-mtx-unique-sol-autonomous-affine*[*OF assms*])
using *has-vderiv-on-sq-mtx-affine* **by** *force*+

lemma *local-flow-sq-mtx-linear*: *local-flow* $((*_V)\ A)$ *UNIV UNIV* $(\lambda t\ s.\ exp\ (t *_R A) *_V s)$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-sq-mtx-affine*[*of* - $\lambda t.\ A\ \lambda t.\ 0$] **apply** *force*
using *has-vderiv-on-sq-mtx-linear*[*of 0*] **by** *auto*

lemma *local-flow-sq-mtx-affine*: *local-flow* $(\lambda s.\ A *_V s + B)$ *UNIV UNIV*
$(\lambda t\ s.\ exp\ (t *_R A) *_V s + exp\ (t *_R A) *_V (\int_0{}^t (exp\ (-\ \tau *_R A) *_V B)\partial\tau))$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-sq-mtx-affine*[*of* - $\lambda t.\ A\ \lambda t.\ B$] **apply** *force*
using *has-vderiv-on-sq-mtx-affine*[*of 0 A*] **by** *auto*

end

0.7.3 Examples

We prove partial correctness specifications of some hybrid systems with our verification components.

theory *hs-vc-examples*
imports *hs-vc-spartan mtx-flows*

begin

Pendulum

The ODEs $x'\ t = y\ t$ and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use *s$1* to represent the x-coordinate and *s$2* for the y-coordinate. We prove that this motion remains circular.

abbreviation *fpend* :: *real^2* ⇒ *real^2* (*f*)
where $f\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ -s\$1)$

abbreviation *pend-flow* :: *real* ⇒ *real^2* ⇒ *real^2* (φ)
where $\varphi\ t\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$1 * cos\ t + s\$2 * sin\ t\ else\ -\ s\$1 * sin\ t + s\$2 * cos\ t)$

— Verified with annotated dynamics.

lemma *pendulum-dyn*: $(\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2) \leq |EVOL\ \varphi\ G\ T]\ (\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2)$
by *force*

— Verified with differential invariants.

lemma *pendulum-inv*: $(\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2) \leq |x\acute{}= f\ \&\ G]\ (\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2)$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified with the flow.

lemma *local-flow-pend*: *local-flow f UNIV UNIV* φ
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *pendulum-flow*: $(\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2) \leq |x\acute{}=f\ \&\ G]\ (\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2)$
by (*force simp*: *local-flow.fbox-g-ode*[*OF local-flow-pend*])

— Verified as a linear system with the flow.

abbreviation *mtx-pend* :: *2 sq-mtx* (*A*)
where $A \equiv$ *mtx*
([*0*, *1*] #
[−*1*, *0*] # [])

lemma *mtx-circ-flow-eq*: $exp\ (t *_R A) *_V s = \varphi\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-sq-mtx-linear*, *symmetric*])
apply(*rule ivp-solsI*, *simp-all add*: *sq-mtx-vec-mult-eq vec-eq-iff*)
unfolding *UNIV-2* **using** *exhaust-2*
by (*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)+

lemma *pendulum-linear*: $(\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2) \leq |x\acute{}=(*_V)\ A\ \&\ G]\ (\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2)$
unfolding *mtx-circ-flow-eq local-flow.fbox-g-ode*[*OF local-flow-sq-mtx-linear*] **by** *auto*

no-notation *fpend* (*f*)
and *pend-flow* (φ)
and *mtx-pend* (*A*)

Bouncing Ball

A ball is dropped from rest at an initial height h. The motion is described with the free-fall equations $x'\ t = v\ t$ and $v'\ t = g$ where g is the constant acceleration due to gravity. The bounce is modelled with a variable assigntment that flips the velocity, thus it is a completely elastic collision with the ground. We use $s\$1$ to ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

abbreviation *fball* :: *real* ⇒ *real^2* ⇒ *real^2* (*f*)
where $f\ g\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ g)$

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real^2* ⇒ *real^2* (φ)

```
  where φ g t s ≡ (χ i. if i = 1 then g * t ^ 2/2 + s$2 * t + s$1 else g * t +
s$2)

— Verified with differential invariants.

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

lemma inv-imp-pos-le[bb-real-arith]:
  assumes 0 > g and inv: 2 * g * x − 2 * g * h = v * v
  shows (x::real) ≤ h
proof−
  have v * v = 2 * g * x − 2 * g * h ∧ 0 > g
    using inv and ‹0 > g› by auto
  hence obs:v * v = 2 * g * (x − h) ∧ 0 > g ∧ v * v ≥ 0
    using left-diff-distrib mult.commute by (metis zero-le-square)
  hence (v * v)/(2 * g) = (x − h)
    by auto
  also from obs have (v * v)/(2 * g) ≤ 0
    using divide-nonneg-neg by fastforce
  ultimately have h − x ≥ 0
    by linarith
  thus ?thesis by auto
qed

lemma bouncing-ball-inv: g < 0 =⇒ h ≥ 0 =⇒
  (λs. s$1 = h ∧ s$2 = 0) ≤
  |LOOP (
    (x´=(f g) & (λ s. s$1 ≥ 0) DINV (λs. 2 * g * s$1 − 2 * g * h − s$2 * s$2
= 0)) ;
    (IF (λ s. s$1 = 0) THEN (2 ::= (λs. − s$2)) ELSE skip))
  INV (λs. 0 ≤ s$1 ∧2 * g * s$1 − 2 * g * h − s$2 * s$2 = 0)]
  (λs. 0 ≤ s$1 ∧ s$1 ≤ h)
  apply(rule fbox-loopI, simp-all, force, force simp: bb-real-arith)
  by (rule fbox-g-odei) (auto intro!: poly-derivatives diff-invariant-rules)

— Verified with annotated dynamics.

lemma inv-conserv-at-ground[bb-real-arith]:
  assumes invar: 2 * g * x = 2 * g * h + v * v
    and pos: g * τ^2 / 2 + v * τ + (x::real) = 0
  shows 2 * g * h + (g * τ + v) * (g * τ + v) = 0
proof−
  from pos have g * τ^2  + 2 * v * τ + 2 * x = 0 by auto
  then have g^2 * τ^2  + 2 * g * v * τ + 2 * g * x = 0
    by (metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right
        monoid-mult-class.power2-eq-square semiring-class.distrib-left)
  hence g^2 * τ^2 + 2 * g * v * τ + v^2 + 2 * g * h = 0
    using invar by (simp add: monoid-mult-class.power2-eq-square)
  hence obs: (g * τ + v)^2 + 2 * g * h = 0
```

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

Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
thus $2 * g * h + (g * \tau + v) * (g * \tau + v) = 0$
by (*simp add*: *add.commute distrib-right power2-eq-square*)
qed

lemma *inv-conserv-at-air*[*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
shows $2 * g * (g * \tau^2 / 2 + v * \tau + (x::real)) =$
$2 * g * h + (g * \tau + v) * (g * \tau + v)$ (**is** *?lhs* = *?rhs*)
proof−
have *?lhs* $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
also have ... $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** *?rhs* $= g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
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-dyn*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$(\lambda s.\ s\$1 = h \wedge s\$2 = 0) \leq$
$|LOOP$ (
$(EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T)$;
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$
$(\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h)$
by (*rule fbox-loopI*) (*auto simp*: *bb-real-arith*)

— Verified with the flow.

lemma *local-flow-ball*: *local-flow* (*f g*) *UNIV UNIV* (φ *g*)
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$(\lambda s.\ s\$1 = h \wedge s\$2 = 0) \leq$
$|LOOP$ (
$(x' = (f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0))$;
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$

INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$)]
($\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h$)
by (*rule fbox-loopI*) (*auto simp*: *bb-real-arith local-flow.fbox-g-ode*[*OF local-flow-ball*])

— Verified as a non-diagonalizable linear system.

abbreviation *mtx-ball* :: *3 sq-mtx* (A)
where $A \equiv$ *mtx* (
[*0*,*1*,*0*] #
[*0*,*0*,*1*] #
[*0*,*0*,*0*] # [])

lemma *pow2-scaleR-mtx-ball*: $(t *_R A)^2 =$ *mtx* (
[*0*,*0*,t^2] #
[*0*,*0*,*0*] #
[*0*,*0*,*0*] # [])
unfolding *power2-eq-square* **apply**(*subst sq-mtx-eq-iff*)
unfolding *sq-mtx-times-eq UNIV-3* **by** *auto*

lemma *powN-scaleR-mtx-ball*: $n > 2 \Longrightarrow (t *_R A)\hat{\ }n = 0$
apply(*induct n*, *simp*, *case-tac* $n \leq 2$)
apply(*subgoal-tac* $n = 2$, *erule ssubst*)
unfolding *power-Suc2 pow2-scaleR-mtx-ball sq-mtx-times-eq UNIV-3*
by (*auto simp*: *sq-mtx-eq-iff*)

lemma *exp-mtx-ball*: *exp* $(t *_R A) = ((t *_R A)^2/_R\ 2) + (t *_R A) + 1$
unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
using *powN-scaleR-mtx-ball* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-mtx-ball-vec-mult-eq*: *exp* $(t *_R A) *_V s =$
vector [$s\$3 * t\hat{\ }2/2 + s\$2 * t + s\$1$, $s\$3 * t + s\$2$, $s\$3$]
apply(*simp add*: *sq-mtx-vec-mult-eq vector-def*)
unfolding *UNIV-3 exp-mtx-ball pow2-scaleR-mtx-ball*
using *exhaust-3* **by** (*auto simp*: *one-mtx3 fun-eq-iff*)

lemma *bouncing-ball-sq-mtx*:
($\lambda s.\ 0 \leq s\$1 \wedge s\$1 = h \wedge s\$2 = 0 \wedge 0 > s\$3$) $\leq$ *fbox*
(*LOOP* (($x' {=} (*_V)\ A$ & ($\lambda\ s.\ s\$1 \geq 0$)) ;
(*IF* ($\lambda\ s.\ s\$1 = 0$) *THEN* (*2* ::= ($\lambda s. - s\$2$)) *ELSE skip*))
INV ($\lambda s.\ 0 {\leq} s\$1 \wedge s\$3{<}0 \wedge 2 * s\$3 * s\$1 = 2 * s\$3 * h + (s\$2 * s\$2)$)))
($\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h$)
apply(*rule fbox-loopI*, *force*, *force simp*: *bb-real-arith*)
apply(*simp add*: *local-flow.fbox-g-ode*[*OF local-flow-sq-mtx-linear*])
unfolding *exp-mtx-ball-vec-mult-eq* **using** *bb-real-arith* **by** *force*

lemma *docking-station-arith*:
assumes (d::*real*) $> x$ **and** $v > 0$
shows $(v = v^2 * t\ /\ (2 * d - 2 * x)) \longleftrightarrow (v * t - v^2 * t^2\ /\ (4 * d - 4 * x) + x = d)$

proof
assume $v = v^2 * t \ / \ (2 * d - 2 * x)$
hence $v * t = 2 * (d - x)$
using *assms* **by** (*simp add*: *eq-divide-eq power2-eq-square*)
hence $v * t - v^2 * t^2 \ / \ (4 * d - 4 * x) + x = 2 * (d - x) - 4 * (d - x)^2 \ / \ (4 * (d - x)) + x$
apply(*subst power-mult-distrib*[*symmetric*])
by (*erule ssubst*, *subst power-mult-distrib*, *simp*)
also have ... $= d$
apply(*simp only*: *mult-divide-mult-cancel-left-if*)
using *assms* **by** (*auto simp*: *power2-eq-square*)
finally show $v * t - v^2 * t^2 \ / \ (4 * d - 4 * x) + x = d$ **.**
next
assume $v * t - v^2 * t^2 \ / \ (4 * d - 4 * x) + x = d$
hence $0 = v^2 * t^2 \ / \ (4 * (d - x)) + (d - x) - v * t$
by *auto*
hence $0 = (4 * (d - x)) * (v^2 * t^2 \ / \ (4 * (d - x)) + (d - x) - v * t)$
by *auto*
also have ... $= v^2 * t^2 + 4 * (d - x)^2 \ - (4 * (d - x)) * (v * t)$
using *assms* **apply**(*simp add*: *distrib-left right-diff-distrib*)
apply(*subst right-diff-distrib*[*symmetric*])+
by (*simp add*: *power2-eq-square*)
also have ... $= (v * t - 2 * (d - x))^2$
by (*simp only*: *power2-diff*, *auto simp*: *field-simps power2-diff*)
finally have $0 = (v * t - 2 * (d - x))^2$ **.**
hence $v * t = 2 * (d - x)$
by *auto*
thus $v = v^2 * t \ / \ (2 * d - 2 * x)$
apply(*subst power2-eq-square*, *subst mult.assoc*)
apply(*erule ssubst*, *subst right-diff-distrib*[*symmetric*])
using *assms* **by** *auto*
qed

Docking station

For a more realistic example of a hybrid system with this vector field, consider a spaceship at initial position x_0 that is approaching a station d at constant velocity $(0::'a) < v_0$. The ship calculates that it needs to deccelerate at $a = - (v_0{}^2 \ / \ ((2::'a) * (d - x_0)))$ in order to anchor itself to the station. As before, we use $s\$1$ for the ship's position, $s\$2$ for its velocity, and $s\$3$ for its acceleration to prove that it will stop moving $s \ \$ \ (2::'b) = (0::'a)$ if and only if its in anchoring position $s \ \$ \ (1::'b) = d$.

— Verified as a non-diagonalizable linear system.

lemma *docking-station*:
assumes $d > x_0$ **and** $v_0 > 0$
shows $(\lambda s.\ s\$1 = x_0 \wedge s\$2 = v_0) \leq$
$|(3 ::= (\lambda s.\ -(v_0\hat{\ }2/(2*(d-x_0))))); x\acute{}=(*_V)\ A\ \&\ G]$

($\lambda s.$ $s\$2 = 0 \longleftrightarrow s\$1 = d$)
apply(*clarsimp simp*: *le-fun-def local-flow.fbox-g-ode*[*OF local-flow-sq-mtx-linear*[*of* *A*]])
unfolding *exp-mtx-ball-vec-mult-eq* **using** *assms* **by** (*simp add*: *docking-station-arith*)

no-notation *fball* (f)
and *ball-flow* (φ)
and *mtx-ball* (A)

Door mechanism

— Verified as a diagonalizable linear system.

abbreviation *mtx-door* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *2 sq-mtx* (A)
where $A\ a\ b \equiv$ *mtx*
([*0*, *1*] #
[a, b] # [])

abbreviation *mtx-chB-door* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *2 sq-mtx* (P)
where $P\ a\ b \equiv$ *mtx*
([a, b] #
[*1*, *1*] # [])

lemma *inv-mtx-chB-door*:
$a \neq b \Longrightarrow (P\ a\ b)^{-1} = (1/(a - b)) *_R$ *mtx*
([*1*, $-b$] #
[-1, a] # [])
apply(*rule sq-mtx-inv-unique*, *unfold scaleR-mtx2 times-mtx2*)
by (*simp add*: *diff-divide-distrib*[*symmetric*] *one-mtx2*)+

lemma *invertible-mtx-chB-door*: $a \neq b \Longrightarrow$ *mtx-invertible* ($P\ a\ b$)
apply(*rule mtx-invertibleI*[*of* - $(P\ a\ b)^{-1}$])
apply(*unfold inv-mtx-chB-door scaleR-mtx2 times-mtx2 one-mtx2*)
by (*subst sq-mtx-eq-iff* , *simp add*: *vector-def frac-diff-eq1*)+

lemma *mtx-door-diagonalizable*:
fixes $a\ b$ :: *real*
defines $\iota_1 \equiv (b - sqrt\ (b\hat{\ }2+4*a))/2$ **and** $\iota_2 \equiv (b + sqrt\ (b\hat{\ }2+4*a))/2$
assumes $b^2 + a * 4 > 0$ **and** $a \neq 0$
shows $A\ a\ b = P\ (-\iota_2/a)\ (-\iota_1/a) * (\text{diag}\ i.\ if\ i = 1\ then\ \iota_1\ else\ \iota_2) * (P\ (-\iota_2/a)\ (-\iota_1/a))^{-1}$
unfolding *assms* **apply**(*subst inv-mtx-chB-door*)
using *assms*(*3*,*4*) **apply**(*simp-all add*: *diag2-eq*[*symmetric*])
unfolding *sq-mtx-times-eq sq-mtx-scaleR-eq UNIV-2* **apply**(*subst sq-mtx-eq-iff*)
using *exhaust-2 assms* **by** (*auto simp*: *field-simps*, *auto simp*: *field-power-simps*)

lemma *mtx-door-solution-eq*:
fixes $a\ b$ :: *real*
defines $\iota_1 \equiv (b - sqrt\ (b^2+4*a))/2$ **and** $\iota_2 \equiv (b + sqrt\ (b^2+4*a))/2$

```
  defines Φ t ≡ mtx (
   [ι2*exp(t*ι1) − ι1*exp(t*ι2),     exp(t*ι2)−exp(t*ι1)]#
   [a*exp(t*ι2) − a*exp(t*ι1), ι2*exp(t*ι2)−ι1*exp(t*ι1)]#[])
  assumes b^2 + a * 4 > 0 and a ≠ 0
   shows P (−ι2/a) (−ι1/a) * (diag i. exp (t * (if i=1 then ι1 else ι2))) * (P
(−ι2/a) (−ι1/a))^−1
   = (1/sqrt (b^2 + a * 4)) *R (Φ t)
  unfolding assms apply(subst inv-mtx-chB-door)
 using assms apply(simp-all add: mtx-times-scaleR-commute, subst sq-mtx-eq-iff)
 unfolding UNIV-2 sq-mtx-times-eq sq-mtx-scaleR-eq sq-mtx-uminus-eq apply(simp-all
add: axis-def)
  by (auto simp: field-simps, auto simp: field-power-simps)+

lemma local-flow-mtx-door:
  fixes a b
  defines ι1 ≡ (b − sqrt (b^2+4*a))/2 and ι2 ≡ (b + sqrt (b^2+4*a))/2
  defines Φ t ≡ mtx (
   [ι2*exp(t*ι1) − ι1*exp(t*ι2),     exp(t*ι2)−exp(t*ι1)]#
   [a*exp(t*ι2) − a*exp(t*ι1), ι2*exp(t*ι2)−ι1*exp(t*ι1)]#[])
  assumes b^2 + a * 4 > 0 and a ≠ 0
  shows local-flow ((*V) (A a b)) UNIV UNIV (λt. (*V) ((1/sqrt (b^2 + a * 4))
*R Φ t))
  unfolding assms using local-flow-sq-mtx-linear[of A a b] assms
 apply(subst (asm) exp-scaleR-diagonal2[OF invertible-mtx-chB-door mtx-door-diagonalizable])
    apply(simp, simp, simp)
  by (subst (asm) mtx-door-solution-eq) simp-all

lemma overdamped-door-arith:
  assumes b^2 + a * 4 > 0 and a < 0 and b ≤ 0 and t ≥ 0 and s1 > 0
  shows 0 ≤ ((b + sqrt (b^2 + 4 * a)) * exp (t * (b − sqrt (b^2 + 4 * a)) / 2) /
2 −
(b − sqrt (b^2 + 4 * a)) * exp (t * (b + sqrt (b^2 + 4 * a)) / 2) / 2) * s1 / sqrt
(b^2 + a * 4)
proof(subst diff-divide-distrib[symmetric], simp)
  have f0: s1 / (2 * sqrt (b^2 + a * 4)) > 0 (is s1/?c3 > 0)
    using assms(1,5) by simp
  have f1: (b − sqrt (b^2 + 4 * a)) < (b + sqrt (b^2 + 4 * a)) (is ?c2 < ?c1)
    and f2: (b + sqrt (b^2 + 4 * a)) < 0
    by (smt assms sqrt-ge-absD real-sqrt-gt-zero)+
  hence f3: exp (t * ?c2 / 2) ≤ exp (t * ?c1 / 2) (is exp ?t1 ≤ exp ?t2)
    using assms(4) by (smt arith-geo-mean-sqrt exp-le-cancel-iff mult-left-mono
        real-sqrt-zero right-diff-distrib times-divide-eq-left)
  hence ?c2 * exp ?t2 ≤ ?c2 * exp ?t1
    using f1 f2 by (smt mult-less-cancel-left)
  also have ... < ?c1 * exp ?t1
    using f1 by auto
  also have... ≤ ?c1 * exp ?t1
    using f1 f2 by auto
  ultimately show 0 ≤ (?c1 * exp ?t1 − ?c2 * exp ?t2) * s1 / ?c3
```

using *f0 f1 assms*(*5*) **by** *auto*
qed

lemma *overdamped-door*:
assumes $b^2 + a * 4 > 0$ **and** $a < 0$ **and** $b \leq 0$ **and** $0 \leq t$
shows $(\lambda s.\ s\$1 = 0) \leq$
$|LOOP$
$(\lambda s.\ \{s.\ s\$1 > 0 \wedge s\$2 = 0\});$
$(x' {=} (*_V)\ (A\ a\ b)\ \&\ G\ on\ \{0..t\}\ UNIV\ @\ 0)$
$INV\ (\lambda s.\ 0 \leq s\$1)]$
$(\lambda s.\ 0 \leq s\ \$\ 1)$
apply(*rule fbox-loopI*, *simp-all add*: *le-fun-def*)
apply(*subst local-flow.fbox-g-ode-ivl*[*OF local-flow-mtx-door*[*OF assms*(*1*)]])
using *assms* **apply**(*simp-all add*: *le-fun-def fbox-def*)
unfolding *sq-mtx-scaleR-eq UNIV-2 sq-mtx-vec-mult-eq*
by (*clarsimp simp*: *overdamped-door-arith*)

no-notation *mtx-door* (*A*)
and *mtx-chB-door* (*P*)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every t minutes, it sets its chronometer to *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -\ a * (T - U)$ where U is $L \geq 0$ when the heater is on, and *0* when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, *3* is the temperature detected by the thermometer, and *4* states whether the heater is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *temp-vec-field* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (*f*)
where $f\ a\ L\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ -\ a * (s\$1 - L)\ else\ 0))$

abbreviation *temp-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (φ)
where $\varphi\ a\ L\ t\ s \equiv (\chi\ i.\ if\ i = 1\ then\ -\ exp(-a * t) * (L - s\$1) + L\ else$
$(if\ i = 2\ then\ t + s\$2\ else\ s\$i))$

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \Longrightarrow \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$
proof(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)
assume *a1*: $0 < a$
have *f2*: $\bigwedge r\ ra.\ |(r::real) + -\ ra| = |ra + -\ r|$
by (*metis abs-minus-commute minus-real-def*)

have $\bigwedge r\ ra\ rb.\ (r::real) * ra + - (r * rb) = r * (ra + - rb)$
by (*metis minus-real-def right-diff-distrib*)
hence $|a * (s_1\$1 + - L) + - (a * (s_2\$1 + - L))| = a * |s_1\$1 + - s_2\$1|$
using *a1* **by** (*simp add*: *abs-mult*)
thus $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$
using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-temp-dyn*:
assumes $0 < (a::real)$
shows *local-lipschitz UNIV UNIV* $(\lambda t::real.\ f\ a\ L)$
apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*)
using *assms*
apply(*simp add*: *norm-diff-temp-dyn*)
apply(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *clarsimp*)
unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-temp*: $a > 0 \Longrightarrow$ *local-flow* $(f\ a\ L)$ *UNIV UNIV* $(\varphi\ a\ L)$
by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-temp-dyn simp*: *forall-4 vec-eq-iff*)

lemma *temp-dyn-down-real-arith*:
assumes $a > 0$ **and** *Thyps*: $0 < Tmin\ Tmin \leq T\ T \leq Tmax$
and *thyps*: $0 \leq (t::real)\ \forall \tau \in \{0..t\}.\ \tau \leq - (ln\ (Tmin\ /\ T)\ /\ a)$
shows $Tmin \leq exp\ (-a * t) * T$ **and** $exp\ (-a * t) * T \leq Tmax$
proof−
have $0 \leq t \wedge t \leq - (ln\ (Tmin\ /\ T)\ /\ a)$
using *thyps* **by** *auto*
hence $ln\ (Tmin\ /\ T) \leq - a * t \wedge - a * t \leq 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have $Tmin\ /\ T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $Tmin\ /\ T \leq exp\ (-a * t)\ exp\ (-a * t) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
thus $Tmin \leq exp\ (-a * t) * T$
using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
show $exp\ (-a * t) * T \leq Tmax$
using *Thyps mult-left-le-one-le*[*OF - exp-ge-zero obs*(*2*), *of T*]
less-eq-real-def order-trans-rules(*23*) **by** *blast*
qed

lemma *temp-dyn-up-real-arith*:
assumes $a > 0$ **and** *Thyps*: $Tmin \leq T\ T \leq Tmax\ Tmax < (L::real)$
and *thyps*: $0 \leq t\ \forall \tau \in \{0..t\}.\ \tau \leq - (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
shows $L - Tmax \leq exp\ (-(a * t)) * (L - T)$
and $L - exp\ (-(a * t)) * (L - T) \leq Tmax$
and $Tmin \leq L - exp\ (-(a * t)) * (L - T)$
proof−

have $0 \leq t \wedge t \leq -(ln\ ((L - Tmax) / (L - T)) / a)$
  using *thyps* **by** *auto*
hence $ln\ ((L - Tmax) / (L - T)) \leq - a * t \wedge - a * t \leq 0$
  using *assms(1) divide-le-cancel* **by** *fastforce*
also have $(L - Tmax) / (L - T) > 0$
  using *Thyps* **by** *auto*
ultimately have $(L - Tmax) / (L - T) \leq exp\ (-a * t) \wedge exp\ (-a * t) \leq 1$
  using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
moreover have $L - T > 0$
  using *Thyps* **by** *auto*
ultimately have *obs*: $(L - Tmax) \leq exp\ (-a * t) * (L - T) \wedge exp\ (-a * t) * (L - T) \leq (L - T)$
  by (*simp add*: *pos-divide-le-eq*)
thus $(L - Tmax) \leq exp\ (-(a * t)) * (L - T)$
  by *auto*
thus $L - exp\ (-(a * t)) * (L - T) \leq Tmax$
  by *auto*
show $Tmin \leq L - exp\ (-(a * t)) * (L - T)$
  using *Thyps* **and** *obs* **by** *auto*
qed

lemmas *fbox-temp-dyn* = *local-flow.fbox-g-ode-ivl*[*OF local-flow-temp - UNIV-I*]

lemma *thermostat*:
assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0) \leq$
|*LOOP*
  — control
  $((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
  $(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
  $(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
  — dynamics
  $(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x' {=} (f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a)$ *on* $\{0..t\}$ *UNIV* @ $0)$
  $ELSE\ (x' {=} (f\ a\ L)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ ((L-Tmax)/(L-s\$3)))/a)$ *on* $\{0..t\}$ *UNIV* @ $0))$)
INV $(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1))]$
$(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax)$
apply(*rule fbox-loopI*, *simp-all add*: *fbox-temp-dyn*[*OF assms*(*1*,*2*)] *le-fun-def*)
using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
  and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of* - *Tmax*] **by** *auto*

no-notation *temp-vec-field* (*f*)
    and *temp-flow* (φ)

Tank

abbreviation *tank-vec-field* :: $real \Rightarrow real\hat{\ }4 \Rightarrow real\hat{\ }4$ (*f*)
where $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

abbreviation *tank-flow* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (φ)
where φ *k* τ *s* ≡ (χ *i*. *if i* = *1 then k* ∗ τ + *s$1 else*
(*if i* = *2 then* τ + *s$2 else s$i*))

abbreviation *tank-guard* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*G*)
where *G Hm k s* ≡ *s$2* ≤ (*Hm* − *s$3*)/*k*

abbreviation *tank-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*I*)
where *I hmin hmax s* ≡ *hmin* ≤ *s$1* ∧ *s$1* ≤ *hmax* ∧ (*s$4* = *0* ∨ *s$4* = *1*)

abbreviation *tank-diff-inv* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*dI*)
where *dI hmin hmax k s* ≡ *s$1* = *k* ∗ *s$2* + *s$3* ∧ *0* ≤ *s$2* ∧
hmin ≤ *s$3* ∧ *s$3* ≤ *hmax* ∧ (*s$4* =*0* ∨ *s$4* = *1*)

lemma *local-flow-tank*: *local-flow* (*f k*) *UNIV UNIV* (φ *k*)
apply (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clarsimp*)
apply(*rule-tac x*=*1/2* **in** *exI*, *clarsimp*, *rule-tac x*=*1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def*, *unfold UNIV-4*)
by (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff*)

lemma *tank-arith*:
assumes *0* ≤ (τ::*real*) **and** *0* < c_o **and** c_o < c_i
shows ∀τ∈{*0*..τ}. τ ≤ − ((*hmin* − *y*) / c_o) ⟹ *hmin* ≤ *y* − c_o ∗ τ
and ∀τ∈{*0*..τ}. τ ≤ (*hmax* − *y*) / (c_i − c_o) ⟹ (c_i − c_o) ∗ τ + *y* ≤ *hmax*
and *hmin* ≤ *y* ⟹ *hmin* ≤ (c_i − c_o) ∗ τ + *y*
and *y* ≤ *hmax* ⟹ *y* − c_o ∗ τ ≤ *hmax*
apply(*simp-all add*: *field-simps le-divide-eq assms*)
using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

lemma *tank-flow*:
assumes *0* ≤ τ **and** *0* < c_o **and** c_o < c_i
shows *I hmin hmax* ≤
|*LOOP*
— control
((*2* ::=(λ*s*.*0*));(*3* ::=(λ*s*. *s$1*));
(*IF* (λ*s*. *s$4* = *0* ∧ *s$3* ≤ *hmin* + *1*) *THEN* (*4* ::= (λ*s*.*1*)) *ELSE*
(*IF* (λ*s*. *s$4* = *1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s*. *s$4* = *0*) *THEN* (*x´*= *f* (c_i−c_o) & *G hmax* (c_i−c_o) *on* {*0*..τ} *UNIV* @ *0*)
ELSE (*x´*= *f* (−c_o) & *G hmin* (−c_o) *on* {*0*..τ} *UNIV* @ *0*))) *INV I hmin hmax*]
I hmin hmax
apply(*rule fbox-loopI*, *simp-all add*: *le-fun-def*)
apply(*clarsimp simp*: *le-fun-def local-flow.fbox-g-ode-ivl*[*OF local-flow-tank assms*(*1*) *UNIV-I*])

using *assms tank-arith*[*OF* - *assms*(*2*,*3*)] **by** *auto*

no-notation *tank-vec-field* (f)
    and *tank-flow* (φ)
    and *tank-loop-inv* (I)
    and *tank-diff-inv* (dI)
    and *tank-guard* (G)

end

0.8 Verification components with predicate transformers

We use the categorical forward box operator $fb_{\mathcal{F}}$ to compute weakest liberal preconditions (wlps) of hybrid programs. Then we repeat the three methods for verifying correctness specifications of the continuous dynamics of a HS.

theory *cat2funcset*
  imports ../*hs-prelims-dyn-sys Transformer-Semantics*.*Kleisli-Quantale*

begin

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

no-notation *bres* (**infixr** $\rightarrow$ *60*)
    and *dagger* (-† [*101*] *100*)
    and *Relation.relcomp* (**infixl** ; *75*)
    and *eta* (η)
    and *kcomp* (**infixl** $\circ_K$ *75*)

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

notation *eta* (*skip*)
  and *kcomp* (**infixl** ; *75*)
  and *g-orbital* ((*1x*´=- & - *on* - - @ -))

0.8.1 Verification of regular programs

Properties of the forward box operator.

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

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

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

unfolding *ffb-eq* **by** *auto*

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

The weakest liberal precondition (wlp) of the "skip" program is the identity.

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

Next, we introduce assignments and their wlps.

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

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

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

The wlp of program composition is just the composition of the wlps.

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

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

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

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

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

lemma *hoare-if-then-else*:
assumes $P \cap \{s.\ T\ s\} \le fb_{\mathcal{F}}\ X\ Q$
and $P \cap \{s.\ \neg\ T\ s\} \le fb_{\mathcal{F}}\ Y\ Q$
shows $P \le fb_{\mathcal{F}}$ (*IF T THEN X ELSE Y*) Q

using *assms* **apply**(*subst ffb-eq*)
apply(*subst* (*asm*) *ffb-eq*)+
unfolding *ifthenelse-def* **by** *auto*

We also deal with finite iteration.

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

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

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

definition *loopi* :: $('a \Rightarrow 'a\ set) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow 'a\ set)$ (*LOOP - INV -* [*64*,*64*] *63*)
where $LOOP\ F\ INV\ I \equiv (kstar\ F)$

lemma *ffb-loopI*: $P \leq \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq Q \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ I\ s\} \Longrightarrow P \leq fb_{\mathcal{F}}\ (LOOP\ F\ INV\ I)\ Q$
unfolding *loopi-def* **using** *ffb-kstarI*[*of P*] **by** *simp*

0.8.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: $(('a{::}ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b\ pred \Rightarrow 'a\ set \Rightarrow ('b \Rightarrow 'b\ set)$ (*EVOL*)
where $EVOL\ \varphi\ G\ T = (\lambda s.\ g\text{-}orbit\ (\lambda t.\ \varphi\ t\ s)\ G\ T)$

lemma *fbox-g-evol*[*simp*]:
fixes $\varphi :: ('a{::}preorder) \Rightarrow 'b \Rightarrow 'b$
shows $fb_{\mathcal{F}}\ (EVOL\ \varphi\ G\ T)\ Q = \{s.\ (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$
unfolding *g-evol-def g-orbit-eq ffb-eq* **by** *auto*

Verification by providing solutions

lemma *ffb-g-orbital*: $fb_{\mathcal{F}}$ $(x\acute{}= f$ & G *on* T S @ $t_0)$ Q =
$\{s. \forall X{\in}Sols\ (\lambda t.\ f)\ T\ S\ t_0\ s. \forall t{\in}T. (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow (X\ t) \in Q\}$
unfolding *ffb-eq g-orbital-eq subset-eq* **by** (*auto simp*: *fun-eq-iff*)

lemma *ffb-g-orbital-eq*: $fb_{\mathcal{F}}$ $(x\acute{}= f$ & G *on* T S @ $t_0)$ Q =
$\{s. \forall X{\in}Sols\ (\lambda t.\ f)\ T\ S\ t_0\ s. \forall t{\in}T. (\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow \mathcal{P}\ X\ (down\ T\ t) \subseteq Q\}$
unfolding *ffb-g-orbital image-le-pred*
apply(*subgoal-tac* $\forall X\ t.\ (\mathcal{P}\ X\ (down\ T\ t) \subseteq Q) = (\forall \tau{\in}down\ T\ t.\ (X\ \tau) \in Q)$)
by (*auto simp*: *image-def*)

context *local-flow*
begin

lemma *ffb-g-ode*: $fb_{\mathcal{F}}$ $(x\acute{}= f$ & G *on* T S @ $0)$ Q =
$\{s.\ s \in S \longrightarrow (\forall t{\in}T. (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$ (**is** - = *?wlp*)
unfolding *ffb-g-orbital* **apply**(*safe*, *clarsimp*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *ballE*)
using *in-ivp-sols* **apply**(*force*, *force*, *force simp*: *init-time ivp-sols-def*)
apply(*subgoal-tac* $\forall \tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ x$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add*: *ivp-sols-def*)
using *init-time* **by** *auto*

lemma *ffb-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow fb_{\mathcal{F}}$ $(x\acute{}{=}f$ & G *on* $\{0..t\}$ S @ $0)$ Q =
$\{s.\ s \in S \longrightarrow (\forall t{\in}\{0..t\}. (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$
unfolding *ffb-g-orbital* **apply**(*clarsimp*, *safe*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *ivp-sols-def*)
apply(*subgoal-tac* $\forall t{\in}\{0..t\}. (\forall \tau{\in}\{0..t\}.\ X\ \tau = \varphi\ \tau\ x)$, *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add*: *ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp*: *closed-segment-eq-real-ivl*)
apply(*force simp*: *closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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

end

Verification with differential invariants

definition *g-ode-inv* :: $(('a{::}banach){\Rightarrow}'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow 'a\ set)$ ((1x´=- & - on - - @ - DINV -))
where $(x\acute{}= f$ & G *on* T S @ t_0 *DINV* $I) = (x\acute{}= f$ & G *on* T S @ $t_0)$

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

lemma *ffb-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ I$ **and** $I \leq Q$
shows $P \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q$
using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
by (*rule ffb-iso*[*OF assms*(*3*)])

lemma *ffb-diff-inv*[*simp*]:
$(\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}) = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
by (*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

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

lemma *bdf-diff-inv*:
$diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G = (bd_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\} \leq \{s.\ I\ s\})$
unfolding *ffb-fbd-galois-var* **by** (*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *diff-inv-guard-ignore*:
assumes $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}$
shows $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}$
using *assms* **unfolding** *ffb-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*
begin

lemma *ffb-diff-inv-eq*: $diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) =$
$(\{s.\ s \in S \longrightarrow I\ s\} = fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ 0)\ \{s.\ s \in S \longrightarrow I\ s\})$
unfolding *ffb-diff-inv*[*symmetric*] *ffb-g-orbital*
using *init-time* **apply**(*auto simp*: *subset-eq ivp-sols-def*)
apply(*subst ivp*(*2*)[*symmetric*], *simp*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *allE*)
using *in-domain has-vderiv-on-domain ivp*(*2*) *init-time* **by** *force*

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

end

lemma *ffb-g-odei*: $P \leq \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.$

$I\ s\} \Longrightarrow$
$\{s.\ I\ s \wedge G\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (x'= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I)\ Q$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $b{=}fb_{\mathcal{F}}\ (x'= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}$ **in** *order.trans*)
apply(*rule-tac* $I{=}\{s.\ I\ s\}$ **in** *ffb-g-orbital-inv*, *simp-all*)
apply(*subst ffb-g-orbital-guard*, *simp*)
by (*rule ffb-iso*, *force*)

0.8.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

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

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

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

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

lemma *ffb-g-orbital-eq-univD*:
assumes $fb_{\mathcal{F}}\ (x'= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ C\ s\} = UNIV$
and $\forall\tau{\in}(down\ T\ t).\ x\ \tau \in (x'= f\ \&\ G\ on\ T\ S\ @\ t_0)\ s$
shows $\forall\tau{\in}(down\ T\ t).\ C\ (x\ \tau)$
proof
fix τ **assume** $\tau \in (down\ T\ t)$
hence $x\ \tau \in (x'= f\ \&\ G\ on\ T\ S\ @\ t_0)\ s$
using *assms*(*2*) **by** *blast*
also have $\forall y.\ y \in (x'= f\ \&\ G\ on\ T\ S\ @\ t_0)\ s \longrightarrow C\ y$
using *assms*(*1*) **unfolding** *ffb-eq* **by** *fastforce*
ultimately show $C\ (x\ \tau)$ **by** *blast*
qed

lemma *diff-cut-axiom*:

assumes *Thyp*: *is-interval* T t_0 ∈ T
and $fb_{\mathcal{F}}$ (x´= f & G *on* T S @ t_0) {s. C s} = *UNIV*
shows $fb_{\mathcal{F}}$ (x´= f & G *on* T S @ t_0) Q = $fb_{\mathcal{F}}$ (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) Q
proof(*rule-tac f*=λ x. $fb_{\mathcal{F}}$ x Q **in** *HOL.arg-cong*, *rule ext*, *rule subset-antisym*)
fix s
{**fix** s' **assume** s' ∈ (x´= f & G *on* T S @ t_0) s
then obtain τ::*real* **and** X **where** *x-ivp*: X ∈ *Sols* (λt. f) T S t_0 s
and X τ = s' **and** τ ∈ T **and** *guard-x*:𝒫 X (*down* T τ) ⊆ {s. G s}
using *g-orbitalD*[*of* s' f G T S t_0 s] **by** *blast*
have ∀ t∈(*down* T τ). 𝒫 X (*down* T t) ⊆ {s. G s}
using *guard-x* **by** (*force simp*: *image-def*)
also have ∀ t∈(*down* T τ). t ∈ T
using ‹τ ∈ T› *Thyp closed-segment-subset-interval* **by** *auto*
ultimately have ∀ t∈(*down* T τ). X t ∈ (x´= f & G *on* T S @ t_0) s
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence ∀ t∈(*down* T τ). C (X t)
using *assms* **unfolding** *ffb-eq* **by** *fastforce*
hence s' ∈ (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) s
using *g-orbitalI*[*OF x-ivp* ‹τ ∈ T›] *guard-x* ‹X τ = s'› **by** *fastforce*}
thus (x´= f & G *on* T S @ t_0) s ⊆ (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) s
by *blast*
next show ⋀s. (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) s ⊆ (x´= f & G *on* T S @ t_0) s
by (*auto simp*: *g-orbital-eq*)
qed

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T t_0 ∈ T
and *ffb-C*: P ≤ $fb_{\mathcal{F}}$ (x´= f & G *on* T S @ t_0) {s. C s}
and *ffb-Q*: P ≤ $fb_{\mathcal{F}}$ (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) Q
shows P ≤ $fb_{\mathcal{F}}$ (x´= f & G *on* T S @ t_0) Q
proof(*subst ffb-eq*, *subst g-orbital-eq*, *clarsimp*)
fix t::*real* **and** X::*real* ⇒ 'a **and** s **assume** s ∈ P **and** t ∈ T
and *x-ivp*:X ∈ *Sols* (λt. f) T S t_0 s
and *guard-x*:∀ τ. *s2p* T τ ∧ τ ≤ t ⟶ G (X τ)
have ∀ r∈(*down* T t). X r ∈ (x´= f & G *on* T S @ t_0) s
using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
hence ∀ t∈(*down* T t). C (X t)
using *ffb-C* ‹s ∈ P› **by** (*subst* (*asm*) *ffb-eq*, *auto*)
hence X t ∈ (x´= f & (λs. G s ∧ C s) *on* T S @ t_0) s
using *guard-x* ‹t ∈ T› **by** (*auto intro*!: *g-orbitalI x-ivp*)
thus (X t) ∈ Q
using ‹s ∈ P› *ffb-Q* **by** (*subst* (*asm*) *ffb-eq*) *auto*
qed

The rules of dL

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

abbreviation *g-global-ode-inv* ::(($'a$::*banach*)$\Rightarrow$$'a$) $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *set*
((1x´=- & - DINV -)) **where** $(x'{=}f\ \&\ G\ DINV\ I) \equiv (x'{=}f\ \&\ G\ on\ UNIV\ UNIV\ @\ 0\ DINV\ I)$

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

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

lemma *DW*: $fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ Q = fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ \{s.\ G\ s \longrightarrow s \in Q\}$
by (*rule diff-weak-axiom*)

lemma *dW*: $\{s.\ G\ s\} \le Q \Longrightarrow P \le fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ Q$
by (*rule diff-weak-rule*)

lemma *DC*:
assumes $fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ \{s.\ C\ s\} = UNIV$
shows $fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ Q = fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ Q$
by (*rule diff-cut-axiom*) (*auto simp*: *assms*)

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

lemma *dI*:
assumes $P \le \{s.\ I\ s\}$ **and** *diff-invariant I f UNIV UNIV 0 G* **and** $\{s.\ I\ s\} \le Q$
shows $P \le fb_{\mathcal{F}}\ (x'{=}\ f\ \&\ G)\ Q$
by (*rule ffb-g-orbital-inv*[*OF assms*(*1*) - *assms*(*3*)]) (*simp add*: *assms*(*2*))

end

0.8.4 Examples

We prove partial correctness specifications of some hybrid systems with our recently described verification components.

theory *cat2funcset-examples*

imports *../mtx-flows cat2funcset*

begin

Preliminary lemmas for the examples.

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

lemma *four-eq-zero*: $(4::4) = 0$
by *simp*

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

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

lemma *UNIV-4*: $(UNIV::4\ set) = \{0,\ 1,\ 2,\ 3\}$
apply *safe* **using** *exhaust-4 four-eq-zero* **by** *auto*

Pendulum

The ODEs $x'\ t = y\ t$ and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use $s\$0$ to represent the x-coordinate and $s\$1$ for the y-coordinate. We prove that this motion remains circular.

— Verified with differential invariants.

abbreviation *fpend* $::$ *real^2* $\Rightarrow$ *real^2* (f)
where $f\ s \equiv (\chi\ i.\ if\ i{=}0\ then\ s\$1\ else\ -s\$0)$

lemma *pendulum-invariants*: $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}}\ (x\acute{}= f\ \&\ G)\ \{s.\ r^2 = (s\$0)^2 + (s\$1)^2\}$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified with the flow.

abbreviation *pend-flow* $::$ *real* $\Rightarrow$ *real^2* $\Rightarrow$ *real^2* (φ)
where $\varphi\ t\ s \equiv (\chi\ i.\ if\ i = 0\ then\ s\$0 \cdot cos\ t + s\$1 \cdot sin\ t\ else\ -\ s\$0 \cdot sin\ t + s\$1 \cdot cos\ t)$

lemma *local-flow-pend*: *local-flow f UNIV UNIV* φ
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
apply(*clarsimp*, *case-tac* $i = 0$, *simp*)
using *exhaust-2 two-eq-zero* **by** (*force intro*!: *poly-derivatives derivative-intros*)+

lemma *pendulum*: $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G)\ \{s.\ r^2 = (s\$0)^2 + (s\$1)^2\}$
by (*force simp*: *local-flow.ffb-g-ode*[*OF local-flow-pend*])

— Verified by providing the dynamics

lemma *pendulum-dyn*: $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}}\ (EVOL\ \varphi\ G\ T)\ \{s.\ r^2 = (s\$0)^2 + (s\$1)^2\}$
by *force*

— Verified as a linear system (using uniqueness).

abbreviation *pend-sq-mtx* :: *2 sq-mtx* (A)
where $A \equiv$ *to-mtx* (χ *i. if i=0 then* e *1 else* $-$ e *0*)

lemma *pend-sq-mtx-exp-eq-flow*: $exp\ (t *_R A) *_V s = \varphi\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-sq-mtx-linear*, *symmetric*])
apply(*rule ivp-solsI*, *clarsimp*)
unfolding *sq-mtx-vec-mult-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *pendulum-sq-mtx*: $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}}\ (x' = (*_V)\ A\ \&\ G)$ $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\}$
unfolding *local-flow.ffb-g-ode*[*OF local-flow-sq-mtx-linear*] *pend-sq-mtx-exp-eq-flow*
by *auto*

no-notation *fpend* (f)
and *pend-sq-mtx* (A)
and *pend-flow* (φ)

Bouncing Ball

A ball is dropped from rest at an initial height h. The motion is described with the free-fall equations $x'\ t = v\ t$ and $v'\ t = g$ where g is the constant acceleration due to gravity. The bounce is modelled with a variable assigntment that flips the velocity, thus it is a completely elastic collision with the ground. We use $s\$0$ to ball's height and $s\$1$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:
assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
shows $(x{::}real) \leq h$
proof–

have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
using *inv* **and** $\langle 0 > g\rangle$ **by** *auto*
hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
hence $(v \cdot v)/(2 \cdot g) = (x - h)$
by *auto*
also from *obs* **have** $(v \cdot v)/(2 \cdot g) \leq 0$
using *divide-nonneg-neg* **by** *fastforce*
ultimately have $h - x \geq 0$
by *linarith*
thus *?thesis* **by** *auto*
qed

abbreviation *fball* :: *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (*f*)
where $f\ g\ s \equiv (\chi\ i.\ if\ i{=}0\ then\ s\$1\ else\ g)$

lemma *bouncing-ball-invariants*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$0 = h \wedge s\$1 = 0\} \leq fb_{\mathcal{F}}$
(*LOOP* (
($x' {=} (f\ g)$ & $(\lambda\ s.\ s\$0 \geq 0)$ *DINV* $(\lambda s.\ 2 \cdot g \cdot s\$0 - 2 \cdot g \cdot h - s\$1 \cdot s\$1 =$
$0))$;
(*IF* $(\lambda\ s.\ s\$0 = 0)$ *THEN* $(1 ::= (\lambda s.\ - s\$1))$ *ELSE skip*))
INV $(\lambda s.\ 0 \leq s\$0 \wedge 2 \cdot g \cdot s\$0 - 2 \cdot g \cdot h - s\$1 \cdot s\$1 = 0))$
$\{s.\ 0 \leq s\$0 \wedge s\$0 \leq h\}$
apply(*rule ffb-loopI*, *simp-all*)
apply(*force*, *force simp*: *bb-real-arith*)
apply(*rule ffb-g-odei*)
by (*auto intro*!: *diff-invariant-rules poly-derivatives simp*: *bb-real-arith*)

— Verified with the flow.

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (φ)
where $\varphi\ g\ t\ s \equiv (\chi\ i.\ if\ i{=}0\ then\ g \cdot t\ \hat{}\ 2/2 + s\$1 \cdot t + s\$0\ else\ g \cdot t + s\$1)$

lemma *local-flow-ball*: *local-flow* $(f\ g)$ *UNIV UNIV* $(\varphi\ g)$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def UNIV-2*)
apply(*clarsimp*, *case-tac* $i = 0$)
using *exhaust-2 two-eq-zero* **by** (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff*)
force

lemma [*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
and *pos*: $g * \tau^2\ /\ 2 + v * \tau + (x{::}real) = 0$
shows $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
proof−
from *pos* **have** $g * \tau^2\ + 2 * v * \tau + 2 * x = 0$ **by** *auto*
then have $g^2 * \tau^2\ + 2 * g * v * \tau + 2 * g * x = 0$

by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right*
monoid-mult-class.power2-eq-square semiring-class.distrib-left)
hence $g^2 * \tau^2 + 2 * g * v * \tau + v^2 + 2 * g * h = 0$
using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
hence *obs*: $(g * \tau + v)^2 + 2 * g * h = 0$
apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
thus $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
by (*simp add*: *add.commute distrib-right 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* $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
by(*auto simp*: *algebra-simps 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*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$0 = h \wedge s\$1 = 0\} \leq fb_{\mathcal{F}}$
(*LOOP* (
$(x' {=} (f\ g)\ \&\ (\lambda\ s.\ s\$0 \geq 0))$;
(*IF* $(\lambda\ s.\ s\$0 = 0)$ *THEN* $(1 ::= (\lambda s.\ -\ s\$1))$ *ELSE skip*))
INV $(\lambda s.\ 0 \leq s\$0 \wedge 2 \cdot g \cdot s\$0 = 2 \cdot g \cdot h + s\$1 \cdot s\$1))$
$\{s.\ 0 \leq s\$0 \wedge s\$0 \leq h\}$
by (*rule ffb-loopI*) (*auto simp*: *bb-real-arith local-flow.ffb-g-ode*[*OF local-flow-ball*])

— Verified by providing the dynamics

lemma *bouncing-ball-dyn*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$0 = h \wedge s\$1 = 0\} \leq fb_{\mathcal{F}}$
(*LOOP* (
(*EVOL* $(\varphi\ g)$ $(\lambda\ s.\ s\$0 \geq 0)$ *T*) ;
(*IF* $(\lambda\ s.\ s\$0 = 0)$ *THEN* $(1 ::= (\lambda s.\ -\ s\$1))$ *ELSE skip*))
INV $(\lambda s.\ 0 \leq s\$0 \wedge 2 \cdot g \cdot s\$0 = 2 \cdot g \cdot h + s\$1 \cdot s\$1))$
$\{s.\ 0 \leq s\$0 \wedge s\$0 \leq h\}$

by (*rule ffb-loopI*) (*auto simp*: *bb-real-arith*)

— Verified as a linear system (computing exponential).

abbreviation *ball-sq-mtx* :: *3 sq-mtx* (*A*)
where *ball-sq-mtx* ≡ *to-mtx* (χ *i*. *if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

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

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

lemma *exp-ball-sq-mtx-simps*:
$exp\ (\tau *_R A)\ \$\$\ 0\ \$\ 0 = 1$ $exp\ (\tau *_R A)\ \$\$\ 0\ \$\ 1 = \tau$ $exp\ (\tau *_R A)\ \$\$\ 0\ \$\ 2$
$= \tau\hat{}2/2$
$exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 0 = 0$ $exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 1 = 1$ $exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 2$
$= \tau$
$exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 0 = 0$ $exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 1 = 0$ $exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 2$
$= 1$
unfolding *exp-ball-sq-mtx scaleR-power ball-sq-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def*
mat-def scaleR-vec-def axis-def plus-vec-def)

lemma *bouncing-ball-sq-mtx*:
$\{s.\ 0 \leq s\$0 \wedge s\$0 = h \wedge s\$1 = 0 \wedge 0 > s\ \$\ 2\} \leq fb_{\mathcal{F}}$
$(LOOP\ ((x' = (*_V)\ A\ \&\ (\lambda\ s.\ s\$0 \geq 0))\ ;$
$(IF\ (\lambda\ s.\ s\$0 = 0)\ THEN\ (1 ::= (\lambda s.\ -\ s\$1))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0{\leq}s\$0 \wedge 0{>}s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot h + (s\$1 \cdot s\$1)))$
$\{s.\ 0 \leq s\$0 \wedge s\$0 \leq h\}$
apply(*rule ffb-loopI*, *simp-all add*: *local-flow.ffb-g-ode*[*OF local-flow-sq-mtx-linear*]
sq-mtx-vec-mult-eq)
apply(*clarsimp*, *force simp*: *bb-real-arith*)
unfolding *UNIV-3* **apply**(*simp add*: *exp-ball-sq-mtx-simps*, *safe*)
using *bb-real-arith*(*2*) **apply**(*force simp*: *add.commute mult.commute*)
using *bb-real-arith*(*3*) **by** (*force simp*: *add.commute mult.commute*)

no-notation *fball* (*f*)
and *ball-flow* (φ)
and *ball-sq-mtx* (*A*)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every t minutes, it sets its chronometer to 0, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = - a * (T - U)$ where U is $L \geq 0$ when the heater is on, and 0 when it is off. We use 0 to denote the room's temperature, 1 is time as measured by the thermostat's chronometer, 2 is the temperature detected by the thermometer, and 3 states whether the heater is on ($s\$3 = 1$) or off ($s\$3 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *temp-vec-field* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (*f*)
  where $f\ a\ L\ s \equiv (\chi\ i.\ \mathit{if}\ i = 1\ \mathit{then}\ 1\ \mathit{else}\ (\mathit{if}\ i = 0\ \mathit{then}\ - a * (s\$0 - L)\ \mathit{else}\ 0))$

abbreviation *temp-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (φ)
  where $\varphi\ a\ L\ t\ s \equiv (\chi\ i.\ \mathit{if}\ i = 0\ \mathit{then}\ - \exp(-a * t) * (L - s\$0) + L\ \mathit{else}$
  $(\mathit{if}\ i = 1\ \mathit{then}\ t + s\$1\ \mathit{else}\ (\mathit{if}\ i = 2\ \mathit{then}\ s\$2\ \mathit{else}\ s\$3)))$

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \Longrightarrow \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$0 - s_2\$0|$
proof(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)
  assume *a1*: $0 < a$
  have *f2*: $\bigwedge r\ ra.\ |(r::real) + - ra| = |ra + - r|$
    by (*metis abs-minus-commute minus-real-def*)
  have $\bigwedge r\ ra\ rb.\ (r::real) * ra + - (r * rb) = r * (ra + - rb)$
    by (*metis minus-real-def right-diff-distrib*)
  hence $|a * (s_1\$0 + - L) + - (a * (s_2\$0 + - L))| = a * |s_1\$0 + - s_2\$0|$
    using *a1* **by** (*simp add*: *abs-mult*)
  thus $|a * (s_2\$0 - L) - a * (s_1\$0 - L)| = a * |s_1\$0 - s_2\$0|$
    using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-temp-dyn*:
  assumes $0 < (a::real)$
  shows *local-lipschitz UNIV UNIV* ($\lambda t::real.\ f\ a\ L$)
  apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
  apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*)
  using *assms* **apply**(*simp-all add*: *norm-diff-temp-dyn*)
  apply(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *clarsimp*)
  unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-temp*: $a > 0 \Longrightarrow$ *local-flow* ($f\ a\ L$) *UNIV UNIV* ($\varphi\ a\ L$)
  by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-temp-dyn*
    simp: *forall-4 vec-eq-iff four-eq-zero*)

```
lemma temp-dyn-down-real-arith:
  assumes a > 0 and Thyps: 0 < Tmin Tmin ≤ T T ≤ Tmax
    and thyps: 0 ≤ (t::real) ∀τ∈{0..t}. τ ≤ − (ln (Tmin / T) / a)
  shows Tmin ≤ exp (−a ∗ t) ∗ T and exp (−a ∗ t) ∗ T ≤ Tmax
proof−
  have 0 ≤ t ∧ t ≤ − (ln (Tmin / T) / a)
    using thyps by auto
  hence ln (Tmin / T) ≤ − a ∗ t ∧ − a ∗ t ≤ 0
    using assms(1) divide-le-cancel by fastforce
  also have Tmin / T > 0
    using Thyps by auto
  ultimately have obs: Tmin / T ≤ exp (−a ∗ t) exp (−a ∗ t) ≤ 1
    using exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less, simp)
  thus Tmin ≤ exp (−a ∗ t) ∗ T
    using Thyps by (simp add: pos-divide-le-eq)
  show exp (−a ∗ t) ∗ T ≤ Tmax
    using Thyps mult-left-le-one-le[OF - exp-ge-zero obs(2), of T]
      less-eq-real-def order-trans-rules(23) by blast
qed

lemma temp-dyn-up-real-arith:
  assumes a > 0 and Thyps: Tmin ≤ T T ≤ Tmax Tmax < (L::real)
    and thyps: 0 ≤ t ∀τ∈{0..t}. τ ≤ − (ln ((L − Tmax) / (L − T)) / a)
  shows L − Tmax ≤ exp (−(a ∗ t)) ∗ (L − T)
    and L − exp (−(a ∗ t)) ∗ (L − T) ≤ Tmax
    and Tmin ≤ L − exp (−(a ∗ t)) ∗ (L − T)
proof−
  have 0 ≤ t ∧ t ≤ − (ln ((L − Tmax) / (L − T)) / a)
    using thyps by auto
  hence ln ((L − Tmax) / (L − T)) ≤ − a ∗ t ∧ − a ∗ t ≤ 0
    using assms(1) divide-le-cancel by fastforce
  also have (L − Tmax) / (L − T) > 0
    using Thyps by auto
  ultimately have (L − Tmax) / (L − T) ≤ exp (−a ∗ t) ∧ exp (−a ∗ t) ≤ 1
    using exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less)
  moreover have L − T > 0
    using Thyps by auto
  ultimately have obs: (L − Tmax) ≤ exp (−a ∗ t) ∗ (L − T) ∧ exp (−a ∗ t)
∗ (L − T) ≤ (L − T)
    by (simp add: pos-divide-le-eq)
  thus (L − Tmax) ≤ exp (−(a ∗ t)) ∗ (L − T)
    by auto
  thus L − exp (−(a ∗ t)) ∗ (L − T) ≤ Tmax
    by auto
  show Tmin ≤ L − exp (−(a ∗ t)) ∗ (L − T)
    using Thyps and obs by auto
qed

lemmas ffb-temp-dyn = local-flow.ffb-g-ode-ivl[OF local-flow-temp - UNIV-I]
```

lemma *thermostat*:
assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $\{s.\ Tmin \leq s\$0 \wedge s\$0 \leq Tmax \wedge s\$3 = 0\} \leq fb_{\mathcal{F}}$
$(LOOP$
— control
$((1 ::= (\lambda s.\ 0));(2 ::= (\lambda s.\ s\$0));$
$(IF\ (\lambda s.\ s\$3 = 0 \wedge s\$2 \leq Tmin + 1)\ THEN\ (3 ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\$3 = 1 \wedge s\$2 \geq Tmax - 1)\ THEN\ (3 ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\$3 = 0)\ THEN\ (x'{=}(f\ a\ 0)\ \&\ (\lambda s.\ s\$1 \leq -\ (ln\ (Tmin/s\$2))/a)$ $on\ \{0..t\}\ UNIV\ @\ 0)$
$ELSE\ (x'{=}(f\ a\ L)\ \&\ (\lambda s.\ s\$1 \leq -\ (ln\ ((L-Tmax)/(L-s\$2)))/a)\ on\ \{0..t\}$ $UNIV\ @\ 0))\ )$
$INV\ (\lambda s.\ Tmin \leq s\$0 \wedge s\$0 \leq Tmax \wedge (s\$3 = 0 \vee s\$3 = 1)))$
$\{s.\ Tmin \leq s\$0 \wedge s\$0 \leq Tmax\}$
apply(*rule ffb-loopI*, *simp-all add*: *ffb-temp-dyn*[*OF assms*(*1*,*2*)] *le-fun-def*, *safe*)
using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of - Tmax*] **by** *auto*

no-notation *temp-vec-field* (f)
and *temp-flow* (φ)

end

0.9 Verification components with Kleene Algebras

We create verification rules based on various Kleene Algebras.

theory *hs-prelims-ka*
imports
KAT-and-DRA.PHL-KAT
KAD.Modal-Kleene-Algebra
Transformer-Semantics.Kleisli-Quantale

begin

0.9.1 Hoare logic and refinement in KAT

Here we derive the rules of Hoare Logic and a refinement calculus in Kleene algebra with tests.

notation t ($\mathfrak{tt}$)

hide-const t

no-notation *ars-r* (r)
and *if-then-else* (*if - then - else - fi* [*64*,*64*,*64*] *63*)
and *while* (*while - do - od* [*64*,*64*] *63*)

context *kat*
begin

— Definitions of Hoare Triple

definition *Hoare* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (H) **where**
$H\ p\ x\ q \longleftrightarrow \mathtt{t}\ p \cdot x \leq x \cdot \mathtt{t}\ q$

lemma *H-consl*: $\mathtt{t}\ p \leq \mathtt{t}\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons1* **by** *blast*

lemma *H-consr*: $\mathtt{t}\ q' \leq \mathtt{t}\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons2* **by** *blast*

lemma *H-cons*: $\mathtt{t}\ p \leq \mathtt{t}\ p' \Longrightarrow \mathtt{t}\ q' \leq \mathtt{t}\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$
by (*simp add*: *H-consl H-consr*)

— Skip program

lemma *H-skip*: $H\ p\ 1\ p$
by (*simp add*: *Hoare-def*)

— Sequential composition

lemma *H-seq*: $H\ p\ x\ r \Longrightarrow H\ r\ y\ q \Longrightarrow H\ p\ (x \cdot y)\ q$
by (*simp add*: *Hoare-def phl-seq*)

— Conditional statement

definition *kat-cond* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*if - then - else - fi* [64,64,64] 63)
where
if p then x else y fi = $(\mathtt{t}\ p \cdot x + n\ p \cdot y)$

lemma *H-var*: $H\ p\ x\ q \longleftrightarrow \mathtt{t}\ p \cdot x \cdot n\ q = 0$
by (*metis Hoare-def n-kat-3 t-n-closed*)

lemma *H-cond-iff*: $H\ p$ (*if r then x else y fi*) $q \longleftrightarrow H\ (\mathtt{t}\ p \cdot \mathtt{t}\ r)\ x\ q \wedge H\ (\mathtt{t}\ p \cdot n\ r)\ y\ q$
proof −
have $H\ p$ (*if r then x else y fi*) $q \longleftrightarrow \mathtt{t}\ p \cdot (\mathtt{t}\ r \cdot x + n\ r \cdot y) \cdot n\ q = 0$
by (*simp add*: *H-var kat-cond-def*)
also have ... $\longleftrightarrow \mathtt{t}\ p \cdot \mathtt{t}\ r \cdot x \cdot n\ q + \mathtt{t}\ p \cdot n\ r \cdot y \cdot n\ q = 0$
by (*simp add*: *distrib-left mult-assoc*)
also have ... $\longleftrightarrow \mathtt{t}\ p \cdot \mathtt{t}\ r \cdot x \cdot n\ q = 0 \wedge \mathtt{t}\ p \cdot n\ r \cdot y \cdot n\ q = 0$
by (*metis add-0-left no-trivial-inverse*)
finally show *?thesis*
by (*metis H-var test-mult*)
qed

lemma *H-cond*: *H* (tt *p* · tt *r*) *x q* $\Longrightarrow$ *H* (tt *p* · *n r*) *y q* $\Longrightarrow$ *H p* (*if r then x else y fi*) *q*
  by (*simp add*: *H-cond-iff*)

— While loop

definition *kat-while* :: *'a* ⇒ *'a* ⇒ *'a* (*while - do - od* [*64*,*64*] *63*) **where**
  while b do x od = (tt *b* · *x*)* · *n b*

definition *kat-while-inv* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (*while - inv - do - od* [*64*,*64*,*64*] *63*) **where**
  while p inv i do x od = *while p do x od*

lemma *H-exp1*: *H* (tt *p* · tt *r*) *x q* $\Longrightarrow$ *H p* (tt *r* · *x*) *q*
  using *Hoare-def n-de-morgan-var2 phl.ht-at-phl-export1* **by** *auto*

lemma *H-while*: *H* (tt *p* · tt *r*) *x p* $\Longrightarrow$ *H p* (*while r do x od*) (tt *p* · *n r*)
proof −
  assume *a1*: *H* (tt *p* · tt *r*) *x p*
  have tt (tt *p* · *n r*) = *n r* · tt *p* · *n r*
    using *n-preserve test-mult* **by** *presburger*
  then show *?thesis*
   using *a1 Hoare-def H-exp1 conway.phl.it-simr phl-export2 kat-while-def* **by** *auto*
qed

lemma *H-while-inv*: tt *p* ≤ tt *i* $\Longrightarrow$ tt *i* · *n r* ≤ tt *q* $\Longrightarrow$ *H* (tt *i* · tt *r*) *x i* $\Longrightarrow$ *H p* (*while r inv i do x od*) *q*
  by (*metis H-cons H-while test-mult kat-while-inv-def*)

— Finite iteration

lemma *H-star*: *H i x i* $\Longrightarrow$ *H i* ($x^\star$) *i*
  unfolding *Hoare-def* **using** *star-sim2* **by** *blast*

lemma *H-star-inv*:
  assumes tt *p* ≤ tt *i* **and** *H i x i* **and** (tt *i*) ≤ (tt *q*)
  shows *H p* ($x^\star$) *q*
proof−
  have *H i* ($x^\star$) *i*
    using *assms*(*2*) *H-star* **by** *blast*
  hence *H p* ($x^\star$) *i*
    unfolding *Hoare-def* **using** *assms*(*1*) *phl-cons1* **by** *blast*
  thus *?thesis*
    unfolding *Hoare-def* **using** *assms*(*3*) *phl-cons2* **by** *blast*
qed

definition *kat-loop-inv* :: *'a* ⇒ *'a* ⇒ *'a* (*loop - inv -* [*64*,*64*] *63*)
  where *loop x inv i* = $x^\star$

lemma *H-loop*: $H\ p\ x\ p \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ p$
  unfolding *kat-loop-inv-def* **by** (*rule H-star*)

lemma *H-loop-inv*: $\mathtt{tt}\ p \le \mathtt{tt}\ i \Longrightarrow H\ i\ x\ i \Longrightarrow \mathtt{tt}\ i \le \mathtt{tt}\ q \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ q$
  unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

lemma *H-inv*: $\mathtt{tt}\ p \le \mathtt{tt}\ i \Longrightarrow \mathtt{tt}\ i \le \mathtt{tt}\ q \Longrightarrow H\ i\ x\ i \Longrightarrow H\ p\ x\ q$
  by (*rule-tac* p'=i **and** q'=i **in** *H-cons*)

lemma *H-inv-plus*: $\mathtt{tt}\ i = i \Longrightarrow \mathtt{tt}\ j = j \Longrightarrow H\ i\ x\ i \Longrightarrow\ H\ j\ x\ j \Longrightarrow\ H\ (i + j)$
$x\ (i + j)$
  unfolding *Hoare-def* **using** *combine-common-factor*
  by (*smt add-commute add.left-commute distrib-left join.sup.absorb-iff1 t-add-closed*)

lemma *H-inv-mult*: $\mathtt{tt}\ i = i \Longrightarrow \mathtt{tt}\ j = j \Longrightarrow H\ i\ x\ i \Longrightarrow\ H\ j\ x\ j \Longrightarrow\ H\ (i \cdot j)$
$x\ (i \cdot j)$
  unfolding *Hoare-def* **by** (*smt n-kat-2 n-mult-comm t-mult-closure mult-assoc*)

end

0.9.2 refinement KAT

class *rkat* = *kat* +
  fixes *Ref* :: $'a \Rightarrow 'a \Rightarrow 'a$
  assumes *spec-def*: $x \le Ref\ p\ q \longleftrightarrow H\ p\ x\ q$

begin

lemma *R1*: $H\ p\ (Ref\ p\ q)\ q$
  using *spec-def* **by** *blast*

lemma *R2*: $H\ p\ x\ q \Longrightarrow x \le Ref\ p\ q$
  by (*simp add*: *spec-def*)

lemma *R-cons*: $\mathtt{tt}\ p \le \mathtt{tt}\ p' \Longrightarrow \mathtt{tt}\ q' \le \mathtt{tt}\ q \Longrightarrow Ref\ p'\ q' \le Ref\ p\ q$
proof –
  assume *h1*: $\mathtt{tt}\ p \le \mathtt{tt}\ p'$ **and** *h2*: $\mathtt{tt}\ q' \le \mathtt{tt}\ q$
  have $H\ p'\ (Ref\ p'\ q')\ q'$
    by (*simp add*: *R1*)
  hence $H\ p\ (Ref\ p'\ q')\ q$
    using *h1 h2 H-consl H-consr* **by** *blast*
  thus *?thesis*
    by (*rule R2*)
qed

— Abort and skip programs

lemma *R-skip*: $1 \leq Ref\ p\ p$
proof –
  have $H\ p\ 1\ p$
    by (*simp add*: *H-skip*)
  thus *?thesis*
    by (*rule R2*)
qed

lemma *R-zero-one*: $x \leq Ref\ 0\ 1$
proof –
  have $H\ 0\ x\ 1$
    by (*simp add*: *Hoare-def*)
  thus *?thesis*
    by (*rule R2*)
qed

lemma *R-one-zero*: $Ref\ 1\ 0 = 0$
proof –
  have $H\ 1\ (Ref\ 1\ 0)\ 0$
    by (*simp add*: *R1*)
  thus *?thesis*
    by (*simp add*: *Hoare-def join.le-bot*)
qed

— Sequential composition

lemma *R-seq*: $(Ref\ p\ r) \cdot (Ref\ r\ q) \leq Ref\ p\ q$
proof –
  have $H\ p\ (Ref\ p\ r)\ r$ **and** $H\ r\ (Ref\ r\ q)\ q$
    by (*simp add*: *R1*)+
  hence $H\ p\ ((Ref\ p\ r) \cdot (Ref\ r\ q))\ q$
    by (*rule H-seq*)
  thus *?thesis*
    by (*rule R2*)
qed

— Conditional statement

lemma *R-cond*: *if v then* $(Ref\ (\mathsf{tt}\ v \cdot \mathsf{tt}\ p)\ q)$ *else* $(Ref\ (n\ v \cdot \mathsf{tt}\ p)\ q)$ *fi* $\leq Ref\ p\ q$
proof –
  have $H\ (\mathsf{tt}\ v \cdot \mathsf{tt}\ p)\ (Ref\ (\mathsf{tt}\ v \cdot \mathsf{tt}\ p)\ q)\ q$ **and** $H\ (n\ v \cdot \mathsf{tt}\ p)\ (Ref\ (n\ v \cdot \mathsf{tt}\ p)$
$q)\ q$
    by (*simp add*: *R1*)+
  hence $H\ p$ (*if v then* $(Ref\ (\mathsf{tt}\ v \cdot \mathsf{tt}\ p)\ q)$ *else* $(Ref\ (n\ v \cdot \mathsf{tt}\ p)\ q)$ *fi*) q
    by (*simp add*: *H-cond n-mult-comm*)
 thus *?thesis*
    by (*rule R2*)
qed

— While loop

lemma *R-while*: *while q do* (*Ref* (tt *p* · tt *q*) *p*) *od* ≤ *Ref p* (tt *p* · *n q*)
proof –
  have *H* (tt *p* · tt *q*) (*Ref* (tt *p* · tt *q*) *p*) *p*
    by (*simp-all add*: *R1*)
  hence *H p* (*while q do* (*Ref* (tt *p* · tt *q*) *p*) *od*) (tt *p* · *n q*)
    by (*simp add*: *H-while*)
  thus *?thesis*
    by (*rule R2*)
qed

— Finite iteration

lemma *R-star*: (*Ref i i*)* ≤ *Ref i i*
proof –
  have *H i* (*Ref i i*) *i*
    using *R1* **by** *blast*
  hence *H i* ((*Ref i i*)*) *i*
    using *H-star* **by** *blast*
  thus *Ref i i** ≤ *Ref i i*
    by (*rule R2*)
qed

lemma *R-loop*: *loop* (*Ref p p*) *inv i* ≤ *Ref p p*
  unfolding *kat-loop-inv-def* **by** (*rule R-star*)

— Invariants

lemma *R-inv*: tt *p* ≤ tt *i* ⟹ tt *i* ≤ tt *q* ⟹ *Ref i i* ≤ *Ref p q*
  using *R-cons* **by** *force*

end

no-notation *kat-cond* (*if - then - else - fi* [*64*,*64*,*64*] *63*)
    and *kat-while* (*while - do - od* [*64*,*64*] *63*)
    and *kat-while-inv* (*while - inv - do - od* [*64*,*64*,*64*] *63*)
    and *kat-loop-inv* (*loop - inv -* [*64*,*64*] *63*)

0.9.3 Verification in AKA (KAD)

Here we derive verification components with weakest liberal preconditions based on antidomain Kleene algebra (or Kleene algebra with domain)

context *antidomain-kleene-algebra*
begin

— Sequential composition

declare *fbox-mult* [*simp*]

— Conditional statement

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

lemma *fbox-export1*: $ad\ p + |x]\ q = |d\ p \cdot x]\ q$
using *a-d-add-closure addual.ars-r-def fbox-def fbox-mult* **by** *auto*

lemma *fbox-cond* [*simp*]: $|$*if p then x else y fi*$]\ q = (ad\ p + |x]\ q) \cdot (d\ p + |y]\ q)$
using *aka-cond-def a-closure′ ads-d-def ans-d-def fbox-add2 fbox-export1* **by** *auto*

— Finite iteration

definition *aka-loop-inv* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*loop - inv -* [*64*,*64*] *63*)
where *loop x inv i* $= x^\star$

lemma *fbox-stari*: $d\ p \le d\ i \Longrightarrow d\ i \le |x]\ i \Longrightarrow d\ i \le d\ q \Longrightarrow d\ p \le |x^\star]\ q$
by (*meson dual-order.trans fbox-iso fbox-star-induct-var*)

lemma *fbox-loopi*: $d\ p \le d\ i \Longrightarrow d\ i \le |x]\ i \Longrightarrow d\ i \le d\ q \Longrightarrow d\ p \le |$*loop x inv i*$]\ q$
unfolding *aka-loop-inv-def* **using** *fbox-stari* **by** *blast*

— Invariants

lemma *fbox-frame*: $d\ p \cdot x \le x \cdot d\ p \Longrightarrow d\ q \le |x]\ r \Longrightarrow d\ p \cdot d\ q \le |x]\ (d\ p \cdot d\ r)$
using *dual.mult-isol-var fbox-add1 fbox-demodalisation3 fbox-simp* **by** *auto*

lemma *plus-inv*: $i \le |x]\ i \Longrightarrow j \le |x]\ j \Longrightarrow (i + j) \le |x]\ (i + j)$
by (*metis ads-d-def dka.dsr5 fbox-simp fbox-subdist join.sup-mono order-trans*)

lemma *mult-inv*: $d\ i \le |x]\ d\ i \Longrightarrow d\ j \le |x]\ d\ j \Longrightarrow (d\ i \cdot d\ j) \le |x]\ (d\ i \cdot d\ j)$
using *fbox-demodalisation3 fbox-frame fbox-simp* **by** *auto*

end

0.9.4 Relational model

We show that relations form Kleene Algebras (KAT and AKA).

interpretation *rel-uq*: *unital-quantale Id* (*O*) $\bigcap$ $\bigcup$ $(\cap)$ $(\subseteq)$ $(\subset)$ $(\cup)$ $\{\}$ *UNIV*
by (*unfold-locales*, *auto*)

lemma *power-is-relpow*: *rel-uq.power X m* $=$ *X ^^ m* **for** *X*::*′a rel*
proof (*induct m*)
case *0* **show** *?case*
by (*metis rel-uq.power-0 relpow.simps*(*1*))

case *Suc* **thus** *?case*
  by (*metis rel-uq.power-Suc2 relpow.simps*(*2*))
qed

lemma *rel-star-def*: $X\hat{}* = (\bigcup m.\ rel\text{-}uq.power\ X\ m)$
  by (*simp add*: *power-is-relpow rtrancl-is-UN-relpow*)

lemma *rel-star-contl*: $X\ O\ Y\hat{}* = (\bigcup m.\ X\ O\ rel\text{-}uq.power\ Y\ m)$
by (*metis rel-star-def relcomp-UNION-distrib*)

lemma *rel-star-contr*: $X\hat{}*\ O\ Y = (\bigcup m.\ (rel\text{-}uq.power\ X\ m)\ O\ Y)$
  by (*metis rel-star-def relcomp-UNION-distrib2*)

interpretation *rel-ka*: *kleene-algebra* (∪) (*O*) *Id* {} (⊆) (⊂) *rtrancl*
proof
  fix *x y z* :: *'a rel*
  show $Id \cup x\ O\ x^* \subseteq x^*$
    by (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)
next
  fix *x y z* :: *'a rel*
  assume $z \cup x\ O\ y \subseteq y$
  thus $x^*\ O\ z \subseteq y$
    by (*simp only*: *rel-star-contr*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductl*)
next
  fix *x y z* :: *'a rel*
  assume $z \cup y\ O\ x \subseteq y$
  thus $z\ O\ x^* \subseteq y$
    by (*simp only*: *rel-star-contl*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductr*)
qed

interpretation *rel-tests*: *test-semiring* (∪) (*O*) *Id* {} (⊆) (⊂) $\lambda x.\ Id \cap (-\ x)$
  by (*standard*, *auto*)

interpretation *rel-kat*: *kat* (∪) (*O*) *Id* {} (⊆) (⊂) *rtrancl* $\lambda x.\ Id \cap (-\ x)$
  by (*unfold-locales*)

definition *rel-R* :: *'a rel* ⇒ *'a rel* ⇒ *'a rel* **where**
  $rel\text{-}R\ P\ Q = \bigcup\{X.\ rel\text{-}kat.Hoare\ P\ X\ Q\}$

interpretation *rel-rkat*: *rkat* (∪) (;) *Id* {} (⊆) (⊂) *rtrancl* ($\lambda X.\ Id \cap -\ X$) *rel-R*
  by (*standard*, *auto simp*: *rel-R-def rel-kat.Hoare-def*)

lemma *RdL-is-rRKAT*: $(\forall x.\ \{(x,x)\};\ R1 \subseteq \{(x,x)\};\ R2) = (R1 \subseteq R2)$
  by *auto*

definition *rel-ad* :: *'a rel* ⇒ *'a rel* **where**
  $rel\text{-}ad\ R = \{(x,x) \mid x.\ \neg\ (\exists y.\ (x,y) \in R)\}$

interpretation *rel-aka*: *antidomain-kleene-algebra rel-ad* (∪) (*O*) *Id* {} (⊆) (⊂)

rtrancl
by *unfold-locales* (*auto simp*: *rel-ad-def*)

0.9.5 State transformer model

We show that state transformers form Kleene Algebras (KAT and AKA).

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

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

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

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

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

definition $0 = \zeta^{\bullet}$

definition *star-nd-fun f* = *qstar f* **for** *f*::'*a nd-fun*

definition $f + g = ((f_{\bullet}) \sqcup (g_{\bullet}))^{\bullet}$

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

lemma *nd-fun-plus-assoc*[*nd-fun-aka*]: $x + y + z = x + (y + z)$
and *nd-fun-plus-comm*[*nd-fun-aka*]: $x + y = y + x$
and *nd-fun-plus-idem*[*nd-fun-aka*]: $x + x = x$ **for** *x*::'*a nd-fun*
unfolding *plus-nd-fun-def* **by** (*simp add*: *ksup-assoc*, *simp-all add*: *ksup-comm*)

lemma *nd-fun-distr*[*nd-fun-aka*]: $(x + y) \cdot z = x \cdot z + y \cdot z$
and *nd-fun-distl*[*nd-fun-aka*]: $x \cdot (y + z) = x \cdot y + x \cdot z$ **for** *x*::'*a nd-fun*
unfolding *plus-nd-fun-def times-nd-fun-def* **by** (*simp-all add*: *kcomp-distr kcomp-distl*)

lemma *nd-fun-plus-zerol*[*nd-fun-aka*]: $0 + x = x$
and *nd-fun-mult-zerol*[*nd-fun-aka*]: $0 \cdot x = 0$
and *nd-fun-mult-zeror*[*nd-fun-aka*]: $x \cdot 0 = 0$ **for** *x*::'*a nd-fun*
unfolding *plus-nd-fun-def zero-nd-fun-def times-nd-fun-def* **by** *auto*

lemma *nd-fun-leq*[*nd-fun-aka*]: $(x \leq y) = (x + y = y)$
and *nd-fun-less*[*nd-fun-aka*]: $(x < y) = (x + y = y \wedge x \neq y)$
and *nd-fun-leq-add*[*nd-fun-aka*]: $z \cdot x \leq z \cdot (x + y)$ **for** *x*::'*a nd-fun*

unfolding *less-eq-nd-fun-def less-nd-fun-def plus-nd-fun-def times-nd-fun-def sup-fun-def*
by (*unfold nd-fun-eq-iff le-fun-def*, *auto simp*: *kcomp-def*)

lemma *nd-star-one*[*nd-fun-aka*]: $1 + x \cdot x^\star \leq x^\star$
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*]: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$ **for** x::'*a nd-fun*
unfolding *plus-nd-fun-def star-nd-fun-def*
apply(*simp-all add*: *fun-star-inductl sup-nd-fun.rep-eq fun-star-inductr*)
by (*metis order-refl sup-nd-fun.rep-eq uwqlka.conway.dagger-unfoldl-eq*)

instance
apply *intro-classes*
using *nd-fun-aka* **by** *simp-all*

end

instantiation *nd-fun* :: (*type*) *kat*
begin

definition $n\ f = (\lambda x.\ if\ ((f_\bullet)\ x = \{\})\ then\ \{x\}\ else\ \{\})^\bullet$

lemma *nd-fun-n-op-one*[*nd-fun-aka*]: $n\ (n\ (1$::'*a nd-fun*$)) = 1$
and *nd-fun-n-op-mult*[*nd-fun-aka*]: $n\ (n\ (n\ x \cdot n\ y)) = n\ x \cdot n\ y$
and *nd-fun-n-op-mult-comp*[*nd-fun-aka*]: $n\ x \cdot n\ (n\ x) = 0$
and *nd-fun-n-op-de-morgan*[*nd-fun-aka*]: $n\ (n\ (n\ x) \cdot n\ (n\ y)) = n\ x + n\ y$ **for** x::'*a nd-fun*
unfolding *n-op-nd-fun-def one-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*

by (*auto simp*: *nd-fun-eq-iff kcomp-def*)

instance
by (*intro-classes*, *auto simp*: *nd-fun-aka*)

end

instantiation *nd-fun* :: (*type*) *rkat*
begin

definition *Ref-nd-fun* $P\ Q \equiv (\lambda s.\ \bigcup\{(f_\bullet)\ s | f.\ Hoare\ P\ f\ Q\})^\bullet$

instance
apply(*intro-classes*)
by (*unfold Hoare-def n-op-nd-fun-def Ref-nd-fun-def times-nd-fun-def*)
(*auto simp*: *kcomp-def le-fun-def less-eq-nd-fun-def*)

end

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

definition *ad f* $= (\lambda x.\ if\ ((f_\bullet)\ x = \{\})\ then\ \{x\}\ else\ \{\})^\bullet$

lemma *nd-fun-ad-zero*[*nd-fun-aka*]: $ad\ x \cdot x = 0$
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*]: $ad\ (ad\ x) + ad\ x = 1$ **for** x::'*a nd-fun*
unfolding *antidomain-op-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*

by (*auto simp*: *nd-fun-eq-iff kcomp-def one-nd-fun-def*)

instance
apply *intro-classes*
using *nd-fun-aka* **by** *simp-all*

end

end

0.10 Verification components with relational MKA

We show that relations form an antidomain Kleene algebra (hence a modal Kleene algebra). We use its forward box operator to derive rules in the algebra for weakest liberal preconditions (wlps) of hybrid programs. Finally, we derive our three methods for verifying correctness specifications for the continuous dynamics of HS in this setting.

theory *mka2rel*
imports ../*hs-prelims-dyn-sys* ../*hs-prelims-ka*

begin

0.10.1 Store and weakest preconditions

type-synonym '*a pred* = '*a* ⇒ *bool*

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
and *Range-Semiring.antirange-semiring-class.ars-r* (*r*)
and *antidomain-semiringl.ads-d* (*d*)
and *n-op* (*n* - [*90*] *91*)
and *Hoare* (*H*)
and *tau* (τ)

notation *Id* (*skip*)
and *zero-class.zero* (*0*)
and *rel-aka.fbox* (*wp*)

definition *p2r* :: '*a pred* ⇒ '*a rel* ((*1*⌈-⌉)) **where**
⌈*P*⌉ = {(*s*,*s*) |*s*. *P s*}

lemma *p2r-simps*[*simp*]:
$\lceil P\rceil \leq \lceil Q\rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
$(\lceil P\rceil = \lceil Q\rceil) = (\forall s.\ P\ s = Q\ s)$
$(\lceil P\rceil\ ;\ \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \wedge Q\ s\rceil$
$(\lceil P\rceil \cup \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \vee Q\ s\rceil$
rel-ad $\lceil P\rceil = \lceil \lambda s.\ \neg\ P\ s\rceil$
rel-aka.ads-d $\lceil P\rceil = \lceil P\rceil$
unfolding *p2r-def rel-ad-def rel-aka.ads-d-def* **by** *auto*

lemma *wp-rel*: *wp* $R\ \lceil P\rceil = \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y\rceil$
unfolding *rel-aka.fbox-def p2r-def rel-ad-def* **by** *auto*

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

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

lemma *wp-assign* [*simp*]: *wp* $(x ::= e)\ \lceil Q\rceil = \lceil \lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)\rceil$
unfolding *wp-rel vec-upd-def assign-def* **by** (*auto simp*: *fun-upd-def*)

abbreviation *cond-sugar* :: $'a$ *pred* $\Rightarrow 'a$ *rel* $\Rightarrow 'a$ *rel* $\Rightarrow 'a$ *rel* (*IF* - *THEN* - *ELSE* - [*64*,*64*] *63*)
where *IF P THEN X ELSE Y* $\equiv$ *rel-aka.aka-cond* $\lceil P\rceil$ *X Y*

abbreviation *loopi-sugar* :: $'a$ *rel* $\Rightarrow 'a$ *pred* $\Rightarrow 'a$ *rel* (*LOOP* - *INV* - [*64*,*64*] *63*)
where *LOOP R INV I* $\equiv$ *rel-aka.aka-loop-inv R* $\lceil I\rceil$

lemma *wp-loopI*: $\lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow \lceil I\rceil \leq$ *wp* $R\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq$ *wp* (*LOOP R INV I*) $\lceil Q\rceil$
using *rel-aka.fbox-loopi*[*of* $\lceil P\rceil$] **by** *auto*

0.10.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: $(('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b$ *pred* $\Rightarrow 'a$ *set* $\Rightarrow 'b$ *rel* (*EVOL*)
where *EVOL* $\varphi\ G\ T = \{(s,s')\ |s\ s'.\ s' \in$ *g-orbit* $(\lambda t.\ \varphi\ t\ s)\ G\ T\}$

lemma *wp-g-dyn*[*simp*]:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows *wp* (*EVOL* φ *G T*) $\lceil Q\rceil = \lceil \lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$
unfolding *wp-rel g-evol-def g-orbit-eq* **by** *auto*

Verification by providing solutions

definition *g-ode* :: $(('a::banach) \Rightarrow 'a) \Rightarrow 'a$ *pred* $\Rightarrow$ *real set* $\Rightarrow 'a$ *set* $\Rightarrow$ *real* $\Rightarrow$
$'a$ *rel* ((*1x´*=*-* & *-* *on* *-* *-* @ *-*))

where $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0) = \{(s,s') \mid s\ s'.\ s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s\}$

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

context *local-flow*
begin

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

lemma *fbox-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow wp\ (x\acute{}{=}f\ \&\ G\ on\ \{0..t\}\ S\ @\ 0)\ \lceil Q\rceil$
$=$
$\lceil\lambda s.\ s \in S \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
unfolding *wp-g-orbital* **apply**(*clarsimp*, *safe*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ s$ **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *ivp-sols-def*)
apply(*subgoal-tac* $\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ X\ \tau = \varphi\ \tau\ s)$, *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add*: *ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp*: *closed-segment-eq-real-ivl*)
apply(*force simp*: *closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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

end

Verification with differential invariants

definition *g-ode-inv* :: $(('a{::}banach){\Rightarrow}'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow$
$real \Rightarrow 'a\ pred \Rightarrow 'a\ rel\ ((1x\acute{}{=}\text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}\ DINV\ \text{-}\ ))$
where $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$

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

lemma *wp-g-orbital-inv*:

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

lemma *wp-diff-inv*[*simp*]: $(\lceil I\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil) = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq wp-g-orbital* **by**(*auto simp*: *p2r-def*)

lemma *diff-inv-guard-ignore*:
assumes $\lceil I\rceil \leq wp\ (x\acute{}= f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$
shows $\lceil I\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$
using *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*
begin

lemma *wp-diff-inv-eq*: $diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) =$
$(\lceil \lambda s.\ s \in S \longrightarrow I\ s\rceil = wp\ (x\acute{}= f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ 0)\ \lceil \lambda s.\ s \in S \longrightarrow I\ s\rceil)$
unfolding *wp-diff-inv*[*symmetric*] *wp-g-orbital*
using *init-time* **apply**(*clarsimp simp*: *ivp-sols-def*)
apply(*safe*, *force*, *force*)
apply(*subst ivp*(*2*)[*symmetric*], *simp*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ s$ **in** *allE*)
using *in-domain has-vderiv-on-domain ivp*(*2*) *init-time* **by** *auto*

lemma *diff-inv-eq-inv-set*:
$diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) = (\forall s.\ I\ s \longrightarrow \gamma^{\varphi}\ s \subseteq \{s.\ I\ s\})$
unfolding *diff-inv-eq-inv-set orbit-def* **by** (*auto simp*: *p2r-def*)

end

lemma *wp-g-odei*: $\lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$\lceil P\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I)\ \lceil Q\rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $b{=}wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$ **in** *order.trans*)
apply(*rule-tac* $I{=}I$ **in** *wp-g-orbital-inv*, *simp-all*)
apply(*subst wp-g-orbital-guard*, *simp*)
by (*rule rel-aka.fbox-iso*, *simp*)

0.10.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

lemma *diff-solve-axiom*:
fixes *c*::′*a*::{*heine-borel*, *banach*}
assumes *0* ∈ *T* **and** *is-interval T open T*
shows *wp* (*x*´=(λ*s*. *c*) & *G on T UNIV* @ *0*) ⌈*Q*⌉ =
⌈λ*s*. ∀ *t*∈*T*. (𝒫 (λ*t*. *s* + *t* $*_R$ *c*) (*down T t*) ⊆ {*s*. *G s*}) ⟶ *Q* (*s* + *t* $*_R$ *c*)⌉
apply(*subst local-flow.wp-g-ode*[**where** *f*=λ*s*. *c* **and** φ=(λ *t x*. *x* + *t* $*_R$ *c*)])
using *line-is-local-flow assms* **by** *auto*

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

lemma *diff-weak-axiom*:
wp (*x*´= *f* & *G on T S* @ t_0) ⌈*Q*⌉ = *wp* (*x*´= *f* & *G on T S* @ t_0) ⌈λ *s*. *G s* ⟶ *Q s*⌉
unfolding *wp-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*:
assumes ⌈*G*⌉ ≤ ⌈*Q*⌉
shows ⌈*P*⌉ ≤ *wp* (*x*´= *f* & *G on T S* @ t_0) ⌈*Q*⌉
using *assms* **apply**(*subst wp-rel*)
by(*auto simp*: *g-orbital-eq g-ode-def*)

lemma *wp-g-evol-IdD*:
assumes *wp* (*x*´= *f* & *G on T S* @ t_0) ⌈*C*⌉ = *Id*
and ∀τ∈(*down T t*). (*s*, *x* τ) ∈ (*x*´= *f* & *G on T S* @ t_0)
shows ∀τ∈(*down T t*). *C* (*x* τ)
proof
fix τ **assume** τ ∈ (*down T t*)
hence *x* τ ∈ *g-orbital f G T S* t_0 *s*
using *assms*(*2*) **unfolding** *g-ode-def* **by** *blast*
also have ∀ *y*. *y* ∈ (*g-orbital f G T S* t_0 *s*) ⟶ *C y*
using *assms*(*1*) **unfolding** *wp-rel g-ode-def* **by**(*auto simp*: *p2r-def*)
ultimately show *C* (*x* τ)
by *blast*
qed

lemma *diff-cut-axiom*:
assumes *Thyp*: *is-interval T* t_0 ∈ *T*
and *wp* (*x*´= *f* & *G on T S* @ t_0) ⌈*C*⌉ = *Id*
shows *wp* (*x*´= *f* & *G on T S* @ t_0) ⌈*Q*⌉ = *wp* (*x*´= *f* & (λ*s*. *G s* ∧ *C s*) *on T S* @ t_0) ⌈*Q*⌉
proof(*rule-tac f*=λ *x*. *wp x* ⌈*Q*⌉ **in** *HOL.arg-cong*, *rule subset-antisym*)
show (*x*´=*f* & *G on T S* @ t_0) ⊆ (*x*´=*f* & λ*s*. *G s* ∧ *C s on T S* @ t_0)
proof(*clarsimp simp*: *g-ode-def*)

fix s **and** s' **assume** $s' \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in$ *Sols* $(\lambda t.\ f)\ T\ S\ t_0\ s$
and $X\ \tau = s'$ **and** $\tau \in T$ **and** *guard-x*:($\mathcal{P}\ X$ (*down* $T\ \tau$) $\subseteq \{s.\ G\ s\}$)
using *g-orbitalD*[*of* $s'\ f\ G\ T\ S\ t_0\ s$] **by** *blast*
have $\forall\, t\in$(*down* $T\ \tau$). $\mathcal{P}\ X$ (*down* $T\ t$) $\subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall\, t\in$(*down* $T\ \tau$). $t \in T$
using ‹$\tau \in T$› *Thyp* **by** *auto*
ultimately have $\forall\, t\in$(*down* $T\ \tau$). $X\ t \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence $\forall\, t\in$(*down* $T\ \tau$). $C\ (X\ t)$
using *wp-g-evol-IdD*[*OF assms*(*3*)] **unfolding** *g-ode-def* **by** *blast*
thus $s' \in$ *g-orbital* f ($\lambda s.\ G\ s \wedge C\ s$) $T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp* ‹$\tau \in T$›] *guard-x* ‹$X\ \tau = s'$› **by** *fastforce*
qed
next show (x´=f & $\lambda s.\ G\ s \wedge C\ s$ *on* $T\ S$ @ t_0) $\subseteq$ (x´=f & G *on* $T\ S$ @ t_0)
by (*auto simp*: *g-orbital-eq g-ode-def*)
qed

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

The rules of dL

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

abbreviation *g-global-ode-inv* :: ((*'a*::*banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a rel* ((*1x*´=- & - *DINV* -)) **where** (x´= f & G *DINV* I) $\equiv$ (x´= f & G *on UNIV UNIV* @ *0 DINV* I)

lemma *DS*:
fixes c::*'a*::{*heine-borel*, *banach*}
shows *wp* (x´=($\lambda s.\ c$) & G) $\lceil Q\rceil$ = $\lceil\lambda x.\ \forall\, t.\ (\forall\,\tau\leq t.\ G\ (x + \tau *_R c)) \longrightarrow Q$ (x

+ $t *_R c)\rceil$
by (*subst diff-solve-axiom*[*of UNIV*]) *auto*

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

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

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

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

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

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

end

0.11 Verification components with MKA and non-deterministic functions

We show that non-deterministic endofunctions form an antidomain Kleene algebra (hence a modal Kleene algebra). We use MKA's forward box operator to derive rules for weakest liberal preconditions (wlps) of hybrid programs. Finally, we derive our three methods for verifying correctness specifications for the continuous dynamics of HS.

theory *mka2ndfun*
imports

../hs-prelims-dyn-sys
../hs-prelims-ka

begin

0.11.1 Store and weakest preconditions

Now that we know that nondeterministic functions form an Antidomain Kleene Algebra, we give a lifting operation from predicates to *'a nd-fun* and use it to compute weakest liberal preconditions.

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

type-synonym *'a pred* = *'a* ⇒ *bool*

notation *fbox* (*wp*)

no-notation *bqtran* (⌊-⌋)
 and *Archimedean-Field.ceiling* (⌈-⌉)
 and *Archimedean-Field.floor* (⌊-⌋)
 and *Relation.relcomp* (**infixl** ; *75*)
 and *Range-Semiring.antirange-semiring-class.ars-r* (r)
 and *antidomain-semiringl.ads-d* (d)
 and *Hoare* (H)
 and *n-op* (n - [*90*] *91*)
 and *tau* (τ)

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

lemma *p2ndf-simps*[*simp*]:
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s.\ P\ s = Q\ s)$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda\ s.\ P\ s \wedge Q\ s \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda\ s.\ P\ s \vee Q\ s \rceil$
 $ad\ \lceil P \rceil = \lceil \lambda s.\ \neg\ P\ s \rceil$
 $d\ \lceil P \rceil = \lceil P \rceil\ \lceil P \rceil \leq \eta^{\bullet}$
 unfolding *less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def ads-d-def*
 by (*auto simp*: *nd-fun-eq-iff kcomp-def le-fun-def antidomain-op-nd-fun-def*)

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

lemma *wp-nd-fun2*: $wp\ (F^{\bullet})\ \lceil P \rceil = \lceil \lambda s.\ \forall s'.\ s' \in (F\ s) \longrightarrow P\ s' \rceil$
 by (*subst wp-nd-fun*, *simp*)

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

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

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

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

We check that *wp* coincides with our other definition of the forward box operator $fb_{\mathcal{F}} = \partial_F \circ bd_{\mathcal{F}} \circ op_K$.

lemma *ffb-is-wp*: $fb_{\mathcal{F}}\ (F_\bullet)\ \{x.\ P\ x\} = \{s.\ \lfloor wp\ F\ \lceil P\rceil\rfloor\ s\}$
  unfolding *ffb-def* **unfolding** *map-dual-def klift-def kop-def fbox-def*
  unfolding *r2f-def f2r-def* **apply** *clarsimp*
  unfolding *antidomain-op-nd-fun-def* **unfolding** *dual-set-def*
  unfolding *times-nd-fun-def kcomp-def* **by** *force*

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

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

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

abbreviation *seq-comp* :: $'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun$ (**infixl** ; *75*)
  where $f\ ;\ g \equiv f \cdot g$

lemma *wp-assign*[*simp*]: $wp\ (x ::= e)\ \lceil Q\rceil = \lceil\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)\rceil$
  unfolding *wp-nd-fun nd-fun-eq-iff vec-upd-def assign-def* **by** *auto*

abbreviation *skip* :: $'a\ nd\text{-}fun$
  where $skip \equiv 1$

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

abbreviation *loopi-sugar* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *nd-fun* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP R INV I* $\equiv$ *aka-loop-inv R* $\lceil I \rceil$

lemma *wp-loopI*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow \lceil I \rceil \leq$ *wp R* $\lceil I \rceil \Longrightarrow \lceil P \rceil \leq$ *wp* (*LOOP R INV I*) $\lceil Q \rceil$
using *fbox-loopi*[*of* $\lceil P \rceil$] **by** *auto*

0.11.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: (($'a$::*ord*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$) $\Rightarrow$ $'b$ *pred* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'b$ *nd-fun* (*EVOL*)
where *EVOL* φ *G T* = ($\lambda s.$ *g-orbit* ($\lambda t.$ φ *t s*) *G T*)$^\bullet$

lemma *wp-g-dyn*[*simp*]:
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
shows *wp* (*EVOL* φ *G T*) $\lceil Q \rceil$ = $\lceil \lambda s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil$
unfolding *wp-nd-fun g-evol-def g-orbit-eq* **by** (*auto simp*: *fun-eq-iff*)

Verification by providing solutions

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

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

context *local-flow*
begin

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

lemma *fbox-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow$ *wp* (x´=*f* & *G on* {*0*..*t*} *S* @ *0*) $\lceil Q \rceil$ =
$\lceil \lambda s.\ s \in S \longrightarrow (\forall t \in \{0..t\}.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \rceil$
unfolding *wp-g-orbital* **apply**(*clarsimp*, *safe*)
apply(*erule-tac x*=$\lambda t.$ φ *t s* **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp*: *closed-segment-eq-real-ivl*)

using *in-ivp-sols-ivl* **apply**(*force simp*: *ivp-sols-def*)
apply(*subgoal-tac* $\forall t \in \{0..t\}$. ($\forall \tau \in \{0..t\}$. $X\ \tau = \varphi\ \tau\ s$), *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add*: *ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp*: *closed-segment-eq-real-ivl*)
apply(*force simp*: *closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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

end

Verification with differential invariants

definition *g-ode-inv* :: $(('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ pred \Rightarrow 'a\ nd\text{-}fun$ ((1x´=- & - on - - @ - DINV -))
where $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)$

lemma *wp-g-orbital-guard*:
assumes $H = (\lambda s.\ G\ s \wedge Q\ s)$
shows $wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil H \rceil$
unfolding *wp-g-orbital* **using** *assms* **by** *auto*

lemma *wp-g-orbital-inv*:
assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
apply(*rule fbox-iso*)
using *assms*(*3*) **by** *auto*

lemma *wp-diff-inv*[*simp*]: $(\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil) = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq wp-g-orbital* **by**(*auto simp*: *fun-eq-iff*)

lemma *diff-inv-guard-ignore*:
assumes $\lceil I \rceil \leq wp\ (x' = f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0)\ \lceil I \rceil$
shows $\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil$
using *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*
begin

lemma *wp-diff-inv-eq*: $diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) =$
$(\lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil = wp\ (x' = f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ 0)\ \lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil)$
unfolding *wp-diff-inv*[*symmetric*] *wp-g-orbital*
using *init-time* **apply**(*clarsimp simp*: *ivp-sols-def*)

apply(*safe*, *force*, *force*)
apply(*subst ivp*(*2*)[*symmetric*], *simp*)
apply(*erule-tac* $x = \lambda t.\ \varphi\ t\ s$ **in** *allE*)
using *in-domain has-vderiv-on-domain ivp*(*2*) *init-time* **by** *auto*

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

end

lemma *wp-g-odei*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil \Longrightarrow$ $\lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I)\ \lceil Q \rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $b = wp\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil$ **in** *order.trans*)
apply(*rule-tac* $I = I$ **in** *wp-g-orbital-inv*, *simp-all*)
apply(*subst wp-g-orbital-guard*, *simp*)
by (*rule fbox-iso*, *simp*)

0.11.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

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

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

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

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

lemma *wp-g-orbit-IdD*:

assumes *wp* $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil = \eta^\bullet$
and $\forall\tau\in(down\ T\ t).\ x\ \tau \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
shows $\forall\tau\in(down\ T\ t).\ C\ (x\ \tau)$
proof
fix τ **assume** $\tau \in (down\ T\ t)$
hence $x\ \tau \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
using *assms*(*2*) **by** *blast*
also have $\forall y.\ y \in$ (*g-orbital* $f\ G\ T\ S\ t_0\ s) \longrightarrow C\ y$
using *assms*(*1*) **unfolding** *wp-nd-fun g-ode-def*
by (*subst* (*asm*) *nd-fun-eq-iff*) *auto*
ultimately show $C\ (x\ \tau)$
by *blast*
qed

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

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and *wp-C*: $\lceil P\rceil \leq$ *wp* $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil$
and *wp-Q*: $\lceil P\rceil \leq$ *wp* $(x\acute{}= f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
shows $\lceil P\rceil \leq$ *wp* $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$

proof(*simp add*: *wp-nd-fun g-orbital-eq g-ode-def*, *clarsimp*)
  fix t::*real* **and** X::*real* ⇒ *'a* **and** s **assume** $P\ s$ **and** $t \in T$
    and *x-ivp*:$X \in$ *ivp-sols* ($\lambda t.\ f$) $T\ S\ t_0\ s$
    and *guard-x*:$\forall x.\ x \in T \wedge x \leq t \longrightarrow G\ (X\ x)$
  have $\forall t \in$(*down* $T\ t$). $X\ t \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
    using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
  hence $\forall t \in$(*down* $T\ t$). $C\ (X\ t)$
    using *wp-C* ‹$P\ s$› **by** (*subst* (*asm*) *wp-nd-fun*, *auto simp*: *g-ode-def*)
  hence $X\ t \in$ *g-orbital* f ($\lambda s.\ G\ s \wedge C\ s$) $T\ S\ t_0\ s$
    using *guard-x* ‹$t \in T$› **by** (*auto intro*!: *g-orbitalI x-ivp*)
  thus $Q\ (X\ t)$
    using ‹$P\ s$› *wp-Q* **by** (*subst* (*asm*) *wp-nd-fun*) (*auto simp*: *g-ode-def*)
qed

The rules of dL

abbreviation *g-global-ode* ::((*'a*::*banach*)⇒*'a*) ⇒ *'a pred* ⇒ *'a nd-fun* ((*1x´*=- & -))
  where ($x´= f$ & G) ≡ ($x´= f$ & G *on UNIV UNIV* @ *0*)

abbreviation *g-global-ode-inv* :: ((*'a*::*banach*)⇒*'a*) ⇒ *'a pred* ⇒ *'a pred* ⇒ *'a nd-fun*
  ((*1x´*=- & - *DINV* -)) **where** ($x´= f$ & G *DINV* I) ≡ ($x´= f$ & G *on UNIV UNIV* @ *0 DINV* I)

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

lemma *solve*:
  assumes *local-flow f UNIV UNIV* φ
    and $\forall s.\ P\ s \longrightarrow (\forall t.\ (\forall \tau \leq t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  shows ⌈P⌉ ≤ *wp* ($x´= f$ & G) ⌈Q⌉
  apply(*rule diff-solve-rule*[*OF assms*(*1*)])
  using *assms*(*2*) **by** *simp*

lemma *DW*: *wp* ($x´= f$ & G) ⌈Q⌉ = *wp* ($x´= f$ & G) ⌈$\lambda s.\ G\ s \longrightarrow Q\ s$⌉
  by (*rule diff-weak-axiom*)

lemma *dW*: ⌈G⌉ ≤ ⌈Q⌉ ⟹ ⌈P⌉ ≤ *wp* ($x´= f$ & G) ⌈Q⌉
  by (*rule diff-weak-rule*)

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

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

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

end

0.11.4 Examples

We prove partial correctness specifications of some hybrid systems with our recently described verification components.

theory *mka-examples*
imports *../mtx-flows mka2rel*

begin

Preliminary preparation for the examples.

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

Pendulum

The ODEs $x'\ t = y\ t$ and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use *s$1* to represent the x-coordinate and *s$2* for the y-coordinate. We prove that this motion remains circular.

abbreviation *fpend* :: *real^2* ⇒ *real^2* (*f*)
where $f\ s \equiv (\chi\ i.\ \mathit{if}\ i = 1\ \mathit{then}\ s\$2\ \mathit{else}\ -s\$1)$

abbreviation *pend-flow* :: *real* ⇒ *real^2* ⇒ *real^2* (φ)
where $\varphi\ t\ s \equiv (\chi\ i.\ \mathit{if}\ i = 1\ \mathit{then}\ s\$1 * \cos t + s\$2 * \sin t$
$\mathit{else}\ -\ s\$1 * \sin t + s\$2 * \cos t)$

— Verified by providing dynamics.

lemma *pendulum-dyn*:
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil \leq wp\ (EVOL\ \varphi\ G\ T)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by *simp*

— Verified with differential invariants.

lemma *pendulum-inv*:
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil \leq wp\ (x\acute{}= f\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

— Verified with the flow.

lemma *local-flow-pend*: *local-flow f UNIV UNIV* φ
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *pendulum-flow*:
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil \leq wp\ (x\acute{}= f\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*simp add*: *local-flow.wp-g-ode*[*OF local-flow-pend*])

— Verified as a linear system (using uniqueness).

abbreviation *pend-sq-mtx* :: *2 sq-mtx* (*A*)
where $A \equiv$ *to-mtx* (χ *i*. *if i=1 then* e *2 else* − e *1*)

lemma *pend-sq-mtx-exp-eq-flow*: $exp\ (t *_R A) *_V s = \varphi\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-sq-mtx-linear*, *symmetric*])
apply(*rule ivp-solsI*, *simp add*: *sq-mtx-vec-mult-def matrix-vector-mult-def*)
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2* **by** (*force simp*: *vec-eq-iff*, *auto*)

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

no-notation *fpend* (*f*)
and *pend-sq-mtx* (*A*)
and *pend-flow* (φ)

Bouncing Ball

A ball is dropped from rest at an initial height h. The motion is described with the free-fall equations $x'\ t = v\ t$ and $v'\ t = g$ where g is the constant acceleration due to gravity. The bounce is modelled with a variable assignment that flips the velocity, thus it is a completely elastic collision with the ground. We use $s\$1$ to ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

abbreviation *fball* :: *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (*f*)
  where *f g s* ≡ (χ *i*. *if i* = *1 then s*\$*2 else g*)

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (φ)
  where φ *g t s* ≡ (χ *i*. *if i* = *1 then g* ∗ *t* ˆ *2*/*2* + *s*\$*2* ∗ *t* + *s*\$*1 else g* ∗ *t* + *s*\$*2*)

— Verified with differential invariants.

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

lemma *inv-imp-pos-le*[*bb-real-arith*]:
  assumes *0* > *g* **and** *inv*: *2* ∗ *g* ∗ *x* − *2* ∗ *g* ∗ *h* = *v* ∗ *v*
  shows (*x*::*real*) ≤ *h*
proof−
  have *v* ∗ *v* = *2* ∗ *g* ∗ *x* − *2* ∗ *g* ∗ *h* ∧ *0* > *g*
    using *inv* **and** ‹*0* > *g*› **by** *auto*
  hence *obs*:*v* ∗ *v* = *2* ∗ *g* ∗ (*x* − *h*) ∧ *0* > *g* ∧ *v* ∗ *v* ≥ *0*
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  hence (*v* ∗ *v*)/(*2* ∗ *g*) = (*x* − *h*)
    by *auto*
  also from *obs* **have** (*v* ∗ *v*)/(*2* ∗ *g*) ≤ *0*
    using *divide-nonneg-neg* **by** *fastforce*
  ultimately have *h* − *x* ≥ *0*
    by *linarith*
  thus *?thesis* **by** *auto*
qed

lemma *bouncing-ball-inv*:
  fixes *h*::*real*
  shows *g* < *0* ⟹ *h* ≥ *0* ⟹ ⌈λ*s*. *s*\$*1* = *h* ∧ *s*\$*2* = *0*⌉ ≤
  wp
    (*LOOP*
      ((*x*´= *f g* & (λ *s*. *s*\$*1* ≥ *0*) *DINV* (λ*s*. *2* ∗ *g* ∗ *s*\$*1* − *2* ∗ *g* ∗ *h* − *s*\$*2* ∗ *s*\$*2* = *0*));
       (*IF* (λ *s*. *s*\$*1* = *0*) *THEN* (*2* ::= (λ*s*. − *s*\$*2*)) *ELSE skip*))
    INV (λ*s*. *0* ≤ *s*\$*1* ∧ *2* ∗ *g* ∗ *s*\$*1* − *2* ∗ *g* ∗ *h* − *s*\$*2* ∗ *s*\$*2* = *0*)
) ⌈λ*s*. *0* ≤ *s*\$*1* ∧ *s*\$*1* ≤ *h*⌉
  apply(*rule wp-loopI*, *simp-all*, *force simp*: *bb-real-arith*)
  by (*rule wp-g-odei*) (*auto intro*!: *poly-derivatives diff-invariant-rules*)

— Verified by providing dynamics.

lemma *inv-conserv-at-ground*[*bb-real-arith*]:
  assumes *invar*: *2* ∗ *g* ∗ *x* = *2* ∗ *g* ∗ *h* + *v* ∗ *v*
    and *pos*: *g* ∗ τ^2 / *2* + *v* ∗ τ + (*x*::*real*) = *0*
  shows *2* ∗ *g* ∗ *h* + (*g* ∗ τ ∗ (*g* ∗ τ + *v*) + *v* ∗ (*g* ∗ τ + *v*)) = *0*
proof−
  from *pos* **have** *g* ∗ τ^2 + *2* ∗ *v* ∗ τ + *2* ∗ *x* = *0* **by** *auto*

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

    Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
thus $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
  by (*simp add*: *monoid-mult-class.power2-eq-square*)
qed

lemma *inv-conserv-at-air*[*bb-real-arith*]:
  assumes *invar*: $2 * g * x = 2 * g * h + v * v$
  shows $2 * g * (g * \tau^2 / 2 + v * \tau + (x{::}real)) =$
  $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
  have *?lhs* $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
    by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
  also have ... $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
    by(*subst invar*, *simp*)
  finally have *?lhs* = *?middle*.
  moreover
  {**have** *?rhs* $= g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
    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-dyn*:
  fixes *h*::*real*
  assumes $g < 0$ **and** $h \geq 0$
  shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
  $\lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil \leq wp$
    (*LOOP*
      ((*EVOL* (φ *g*) ($\lambda s.\ 0 \leq s\$1$) *T*);
      (*IF* (λ *s*. $s\$1 = 0$) *THEN* (*2* ::= ($\lambda s.\ - s\$2$)) *ELSE skip*))
    INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$))
  $\lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
  by (*rule wp-loopI*) (*auto simp*: *bb-real-arith*)

— Verified with the flow.

lemma *local-flow-ball*: *local-flow* (*f g*) *UNIV UNIV* (φ *g*)
  apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*,
clarsimp)

apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *bouncing-ball-flow*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$
(*LOOP*
$((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
apply(*rule wp-loopI*, *simp-all add*: *local-flow.wp-g-ode*[*OF local-flow-ball*])
by (*auto simp*: *bb-real-arith*)

— Verified as a linear system (computing exponential).

abbreviation *ball-sq-mtx* :: *3 sq-mtx* (*A*)
where *ball-sq-mtx* $\equiv$ *to-mtx* (χ *i*. *if* $i = 1$ *then* e *2 else if* $i = 2$ *then* e *3 else 0*)

lemma *ball-sq-mtx-pow2*: $A^2 =$ *to-mtx* (χ *i*. *if* $i = 1$ *then* e *3 else 0*)
unfolding *monoid-mult-class.power2-eq-square times-sq-mtx-def*
by (*simp add*: *to-mtx-inject vec-eq-iff matrix-matrix-mult-def*)

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

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

lemma *exp-ball-sq-mtx-simps*:
$exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 1 = 1\ exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 2 = \tau\ exp\ (\tau *_R A)\ \$\$\ 1\ \$\ 3 = \tau\hat{}2/2$
$exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 1 = 0\ exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 2 = 1\ exp\ (\tau *_R A)\ \$\$\ 2\ \$\ 3 = \tau$
$exp\ (\tau *_R A)\ \$\$\ 3\ \$\ 1 = 0\ exp\ (\tau *_R A)\ \$\$\ 3\ \$\ 2 = 0\ exp\ (\tau *_R A)\ \$\$\ 3\ \$\ 3 = 1$
unfolding *exp-ball-sq-mtx scaleR-power ball-sq-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def*
mat-def scaleR-vec-def axis-def plus-vec-def)

lemma *bouncing-ball-sq-mtx*:
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 = h \wedge s\$2 = 0 \wedge 0 > s\$3 \rceil \leq wp$

(*LOOP*
((*x*´=($*_V$)*A* & (λ *s*. *s*\$*1* $\geq$ *0*));
(*IF* (λ *s*. *s*\$*1* = *0*) *THEN* (*2* ::= (λ*s*. − *s*\$*2*)) *ELSE skip*))
INV (λ*s*. *0* $\leq$ *s*\$*1* $\wedge$ *0* > *s*\$*3* $\wedge$ *2* $\cdot$ *s*\$*3* $\cdot$ *s*\$*1* = *2* $\cdot$ *s*\$*3* $\cdot$ *h* + (*s*\$*2* $\cdot$ *s*\$*2*)))
$\lceil$$\lambda$*s*. *0* $\leq$ *s*\$*1* $\wedge$ *s*\$*1* $\leq$ *h*$\rceil$
apply(*rule wp-loopI*, *simp-all add*: *local-flow.wp-g-ode*[*OF local-flow-sq-mtx-linear*])
apply(*force simp*: *bb-real-arith*)
apply(*simp add*: *sq-mtx-vec-mult-eq*)
unfolding *UNIV-3* **apply**(*simp add*: *exp-ball-sq-mtx-simps*, *safe*)
using *bb-real-arith*(*2*) **apply**(*force simp*: *add.commute mult.commute*)
using *bb-real-arith*(*3*) **by** (*force simp*: *add.commute mult.commute*)

no-notation *fball* (*f*)
and *ball-flow* (φ)
and *ball-sq-mtx* (*A*)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every *t* minutes, it sets its chronometer to *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -\ a * (T - U)$ where *U* is $L \geq 0$ when the heater is on, and *0* when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, *3* is the temperature detected by the thermometer, and *4* states whether the heater is on (*s*\$*4* = *1*) or off (*s*\$*4* = *0*). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *temp-vec-field* :: *real* ⇒ *real* ⇒ *real*ˆ*4* ⇒ *real*ˆ*4* (*f*)
where *f a L s* ≡ (χ *i*. *if i* = *2 then 1 else* (*if i* = *1 then* − *a* ∗ (*s*\$*1* − *L*) *else* *0*))

abbreviation *temp-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real*ˆ*4* ⇒ *real*ˆ*4* (φ)
where φ *a L t s* ≡ (χ *i*. *if i* = *1 then* − *exp*(−*a* ∗ *t*) ∗ (*L* − *s*\$*1*) + *L else* (*if i* = *2 then t* + *s*\$*2 else s*\$*i*))

— Verified with the flow.

lemma *norm-diff-temp-dyn*: *0* < *a* ⟹ ‖*f a L* s_1 − *f a L* s_2‖ = |*a*| ∗ |s_1\$*1* − s_2\$*1*|
proof(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)
assume *a1*: *0* < *a*
have *f2*: ⋀*r ra*. |(*r*::*real*) + − *ra*| = |*ra* + − *r*|
by (*metis abs-minus-commute minus-real-def*)
have ⋀*r ra rb*. (*r*::*real*) ∗ *ra* + − (*r* ∗ *rb*) = *r* ∗ (*ra* + − *rb*)
by (*metis minus-real-def right-diff-distrib*)
hence |*a* ∗ (s_1\$*1* + − *L*) + − (*a* ∗ (s_2\$*1* + − *L*))| = *a* ∗ |s_1\$*1* + − s_2\$*1*|
using *a1* **by** (*simp add*: *abs-mult*)
thus |*a* ∗ (s_2\$*1* − *L*) − *a* ∗ (s_1\$*1* − *L*)| = *a* ∗ |s_1\$*1* − s_2\$*1*|

using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-temp-dyn*:
assumes *0* < (*a*::*real*)
shows *local-lipschitz UNIV UNIV* (λ*t*::*real*. *f a L*)
apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
apply(*clarsimp*, *rule-tac x*=*1* **in** *exI*, *clarsimp*, *rule-tac x*=*a* **in** *exI*)
using *assms*
apply(*simp-all add*: *norm-diff-temp-dyn*)
apply(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *clarsimp*)
unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-temp*: *a* > *0* ⟹ *local-flow* (*f a L*) *UNIV UNIV* (φ *a L*)
by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-temp-dyn simp*: *forall-4 vec-eq-iff*)

lemma *temp-dyn-down-real-arith*:
assumes *a* > *0* **and** *Thyps*: *0* < *Tmin Tmin* ≤ *T T* ≤ *Tmax*
and *thyps*: *0* ≤ (*t*::*real*) ∀τ∈{*0*..*t*}. τ ≤ − (*ln* (*Tmin* / *T*) / *a*)
shows *Tmin* ≤ *exp* (−*a* ∗ *t*) ∗ *T* **and** *exp* (−*a* ∗ *t*) ∗ *T* ≤ *Tmax*
proof−
have *0* ≤ *t* ∧ *t* ≤ − (*ln* (*Tmin* / *T*) / *a*)
using *thyps* **by** *auto*
hence *ln* (*Tmin* / *T*) ≤ − *a* ∗ *t* ∧ − *a* ∗ *t* ≤ *0*
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have *Tmin* / *T* > *0*
using *Thyps* **by** *auto*
ultimately have *obs*: *Tmin* / *T* ≤ *exp* (−*a* ∗ *t*) *exp* (−*a* ∗ *t*) ≤ *1*
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
thus *Tmin* ≤ *exp* (−*a* ∗ *t*) ∗ *T*
using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
show *exp* (−*a* ∗ *t*) ∗ *T* ≤ *Tmax*
using *Thyps mult-left-le-one-le*[*OF* - *exp-ge-zero obs*(*2*), *of T*]
less-eq-real-def order-trans-rules(*23*) **by** *blast*
qed

lemma *temp-dyn-up-real-arith*:
assumes *a* > *0* **and** *Thyps*: *Tmin* ≤ *T T* ≤ *Tmax Tmax* < (*L*::*real*)
and *thyps*: *0* ≤ *t* ∀τ∈{*0*..*t*}. τ ≤ − (*ln* ((*L* − *Tmax*) / (*L* − *T*)) / *a*)
shows *L* − *Tmax* ≤ *exp* (−(*a* ∗ *t*)) ∗ (*L* − *T*)
and *L* − *exp* (−(*a* ∗ *t*)) ∗ (*L* − *T*) ≤ *Tmax*
and *Tmin* ≤ *L* − *exp* (−(*a* ∗ *t*)) ∗ (*L* − *T*)
proof−
have *0* ≤ *t* ∧ *t* ≤ − (*ln* ((*L* − *Tmax*) / (*L* − *T*)) / *a*)
using *thyps* **by** *auto*
hence *ln* ((*L* − *Tmax*) / (*L* − *T*)) ≤ − *a* ∗ *t* ∧ − *a* ∗ *t* ≤ *0*
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have (*L* − *Tmax*) / (*L* − *T*) > *0*

using *Thyps* **by** *auto*
ultimately have $(L - Tmax) / (L - T) \leq exp\ (-a * t) \wedge exp\ (-a * t) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
moreover have $L - T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $(L - Tmax) \leq exp\ (-a * t) * (L - T) \wedge exp\ (-a * t) * (L - T) \leq (L - T)$
by (*simp add*: *pos-divide-le-eq*)
thus $(L - Tmax) \leq exp\ (-(a * t)) * (L - T)$
by *auto*
thus $L - exp\ (-(a * t)) * (L - T) \leq Tmax$
by *auto*
show $Tmin \leq L - exp\ (-(a * t)) * (L - T)$
using *Thyps* **and** *obs* **by** *auto*
qed

lemmas *fbox-temp-dyn* = *local-flow.fbox-g-ode-ivl*[*OF local-flow-temp - UNIV-I*]

lemma *thermostat*:
assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$
(*LOOP*
— control
$((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x' {=} (f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a)\ on\ \{0..t\}\ UNIV\ @\ 0)$
$ELSE\ (x' {=} (f\ a\ L)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ ((L{-}Tmax)/(L{-}s\$3)))/a)\ on\ \{0..t\}\ UNIV\ @\ 0))\ )$
$INV\ (\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
$\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \rceil$
apply(*rule wp-loopI*, *simp-all add*: *fbox-temp-dyn*[*OF assms*(*1*,*2*)])
using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of* - *Tmax*] **by** *auto*

no-notation *temp-vec-field* (f)
and *temp-flow* (φ)

end

0.12 Verification and refinement of HS in the relational KAT

We use our relational model to obtain verification and refinement components for hybrid programs. We devise three methods for reasoning with evolution commands and their continuous dynamics: providing flows, solutions or invariants.

theory *kat2rel*
  imports
    ../hs-prelims-ka
    ../hs-prelims-dyn-sys

begin

0.12.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

no-notation *Archimedean-Field.ceiling* ($\lceil\text{-}\rceil$)
  and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor\text{-}\rfloor$)
  and *tau* (τ)
  and *proto-near-quantale-class.bres* (**infixr** $\rightarrow$ *60*)

notation *Id* (*skip*)

— Canonical lifting from predicates to relations and its simplification rules

definition *p2r* :: $'a\ pred \Rightarrow 'a\ rel$ ($\lceil\text{-}\rceil$) **where**
  $\lceil P\rceil = \{(s,s)\ |s.\ P\ s\}$

lemma *p2r-simps*[*simp*]:
  $\lceil P\rceil \leq \lceil Q\rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
  $(\lceil P\rceil = \lceil Q\rceil) = (\forall s.\ P\ s = Q\ s)$
  $(\lceil P\rceil\ ;\ \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \wedge Q\ s\rceil$
  $(\lceil P\rceil \cup \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \vee Q\ s\rceil$
  $rel\text{-}tests.t\ \lceil P\rceil = \lceil P\rceil$
  $(-\ Id) \cup \lceil P\rceil = -\ \lceil \lambda s.\ \neg\ P\ s\rceil$
  $Id \cap (-\ \lceil P\rceil) = \lceil \lambda s.\ \neg\ P\ s\rceil$
  unfolding *p2r-def* **by** *auto*

— Meaning of the relational hoare triple

lemma *rel-kat-H*: $rel\text{-}kat.Hoare\ \lceil P\rceil\ X\ \lceil Q\rceil \longleftrightarrow (\forall s\ s'.\ P\ s \longrightarrow (s,s') \in X \longrightarrow Q\ s')$
  by (*simp add*: *rel-kat.Hoare-def*, *auto simp add*: *p2r-def*)

— Hoare triple for skip and a simp-rule

lemma *H-skip*: $rel\text{-}kat.Hoare\ \lceil P\rceil\ skip\ \lceil P\rceil$
  using *rel-kat.H-skip* **by** *blast*

lemma *sH-skip*[*simp*]: $rel\text{-}kat.Hoare\ \lceil P\rceil\ skip\ \lceil Q\rceil \longleftrightarrow \lceil P\rceil \leq \lceil Q\rceil$
  unfolding *rel-kat-H* **by** *simp*

— We introduce assignments and compute derive their rule of Hoare logic.

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

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

lemma *H-assign*: $P = (\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow rel\text{-}kat.Hoare\ \lceil P\rceil$ $(x ::= e)\ \lceil Q\rceil$
  unfolding *rel-kat-H assign-def vec-upd-def* **by** *force*

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

— Next, the Hoare rule of the composition

lemma *H-seq*: $rel\text{-}kat.Hoare\ \lceil P\rceil\ X\ \lceil R\rceil \Longrightarrow rel\text{-}kat.Hoare\ \lceil R\rceil\ Y\ \lceil Q\rceil \Longrightarrow rel\text{-}kat.Hoare$ $\lceil P\rceil\ (X\ ;\ Y)\ \lceil Q\rceil$
  by (*auto intro*: *rel-kat.H-seq*)

lemma *sH-seq*:
  $rel\text{-}kat.Hoare\ \lceil P\rceil\ (X\ ;\ Y)\ \lceil Q\rceil = rel\text{-}kat.Hoare\ \lceil P\rceil\ (X)\ \lceil\lambda s.\ \forall s'.\ (s,\ s') \in Y$ $\longrightarrow Q\ s'\rceil$
  unfolding *rel-kat-H* **by** *auto*

— Rewriting the Hoare rule for the conditional statement

abbreviation *cond-sugar* :: $'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ (*IF - THEN - ELSE -* [64,64] 63)
  where *IF B THEN X ELSE Y* $\equiv$ *rel-kat.kat-cond* $\lceil B\rceil$ *X Y*

lemma *H-cond*: $rel\text{-}kat.Hoare\ \lceil P \sqcap B\rceil\ X\ \lceil Q\rceil \Longrightarrow rel\text{-}kat.Hoare\ \lceil P \sqcap -\ B\rceil\ Y$ $\lceil Q\rceil \Longrightarrow$
  $rel\text{-}kat.Hoare\ \lceil P\rceil$ (*IF B THEN X ELSE Y*) $\lceil Q\rceil$
  by (*rule rel-kat.H-cond*, *auto simp*: *rel-kat-H*)

lemma *sH-cond*[*simp*]: $rel\text{-}kat.Hoare\ \lceil P\rceil$ (*IF B THEN X ELSE Y*) $\lceil Q\rceil \longleftrightarrow$
  $(rel\text{-}kat.Hoare\ \lceil P \sqcap B\rceil\ X\ \lceil Q\rceil \wedge rel\text{-}kat.Hoare\ \lceil P \sqcap -\ B\rceil\ Y\ \lceil Q\rceil)$
  by (*auto simp*: *rel-kat.H-cond-iff rel-kat-H*)

— Rewriting the Hoare rule for the while loop

abbreviation *while-inv-sugar* :: $'a\ pred \Rightarrow 'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ (*WHILE - INV - DO -* [64,64,64] 63)
  where *WHILE B INV I DO X* $\equiv$ *rel-kat.kat-while-inv* $\lceil B\rceil\ \lceil I\rceil$ *X*

lemma *sH-while-inv*: $\forall s.\ P\ s \longrightarrow I\ s \Longrightarrow \forall s.\ I\ s \wedge \neg\ B\ s \longrightarrow Q\ s \Longrightarrow$ *rel-kat.Hoare* $\lceil I \sqcap B\rceil$ X $\lceil I\rceil$
$\Longrightarrow$ *rel-kat.Hoare* $\lceil P\rceil$ (*WHILE B INV I DO X*) $\lceil Q\rceil$
by (*rule rel-kat.H-while-inv*, *auto simp*: *p2r-def rel-kat.Hoare-def*, *fastforce*)

— Finally, we add a Hoare triple rule for finite iterations.

abbreviation *loopi-sugar* :: *'a rel* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a rel* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP X INV I* $\equiv$ *rel-kat.kat-loop-inv X* $\lceil I\rceil$

lemma *H-loop*: *rel-kat.Hoare* $\lceil P\rceil$ X $\lceil P\rceil$ $\Longrightarrow$ *rel-kat.Hoare* $\lceil P\rceil$ (*LOOP X INV I*) $\lceil P\rceil$
by (*auto intro*: *rel-kat.H-loop*)

lemma *H-loopI*: *rel-kat.Hoare* $\lceil I\rceil$ X $\lceil I\rceil$ $\Longrightarrow$ $\lceil P\rceil \subseteq \lceil I\rceil$ $\Longrightarrow$ $\lceil I\rceil \subseteq \lceil Q\rceil$ $\Longrightarrow$ *rel-kat.Hoare* $\lceil P\rceil$ (*LOOP X INV I*) $\lceil Q\rceil$
using *rel-kat.H-loop-inv*[*of* $\lceil P\rceil$ $\lceil I\rceil$ X $\lceil Q\rceil$] **by** *auto*

0.12.2 Verification of hybrid programs

— Verification by providing evolution

definition *g-evol* :: ((*'a::ord*) $\Rightarrow$ *'b* $\Rightarrow$ *'b*) $\Rightarrow$ *'b pred* $\Rightarrow$ *'a set* $\Rightarrow$ *'b rel* (*EVOL*)
where *EVOL* φ G T = $\{(s,s')\ |s\ s'.\ s' \in$ *g-orbit* $(\lambda t.\ \varphi\ t\ s)\ G\ T\}$

lemma *H-g-evol*:
fixes φ :: (*'a::preorder*) $\Rightarrow$ *'b* $\Rightarrow$ *'b*
assumes $P = (\lambda s.\ (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows *rel-kat.Hoare* $\lceil P\rceil$ (*EVOL* φ G T) $\lceil Q\rceil$
unfolding *rel-kat-H g-evol-def g-orbit-eq* **using** *assms* **by** *clarsimp*

lemma *sH-g-evol*[*simp*]:
fixes φ :: (*'a::preorder*) $\Rightarrow$ *'b* $\Rightarrow$ *'b*
shows *rel-kat.Hoare* $\lceil P\rceil$ (*EVOL* φ G T) $\lceil Q\rceil$ = $(\forall s.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
unfolding *rel-kat-H g-evol-def g-orbit-eq* **by** *auto*

— Verification by providing solutions

definition *g-ode* :: ((*'a::banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *real set* $\Rightarrow$ *'a set* $\Rightarrow$ *real* $\Rightarrow$ *'a rel* ((*1x´=- & - on - - @ -*))
where ($x´= f$ & G *on* T S @ t_0) = $\{(s,s')\ |s\ s'.\ s' \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s\}$

lemma *H-g-orbital*:
$P = (\lambda s.\ (\forall X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \Longrightarrow$
rel-kat.Hoare $\lceil P\rceil$ ($x´= f$ & G *on* T S @ t_0) $\lceil Q\rceil$
unfolding *rel-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

lemma *sH-g-orbital*: *rel-kat.Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ t_0)$ $\lceil Q \rceil$ =
$(\forall s. \ P \ s \longrightarrow (\forall X \in ivp\text{-}sols \ (\lambda t. \ f) \ T \ S \ t_0 \ s. \ \forall t \in T. \ (\forall \tau \in down \ T \ t. \ G \ (X \ \tau))$
$\longrightarrow Q \ (X \ t)))$
unfolding *g-orbital-eq g-ode-def rel-kat-H* **by** *auto*

context *local-flow*
begin

lemma *H-g-ode*:
assumes $P = (\lambda s. \ s \in S \longrightarrow (\forall t \in T. \ (\forall \tau \in down \ T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$
shows *rel-kat.Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ 0)$ $\lceil Q \rceil$
proof(*unfold rel-kat-H g-ode-def g-orbital-eq assms*, *clarsimp*)
fix *s t X*
assume *hyps*: $t \in T$ $\forall x. \ x \in T \land x \le t \longrightarrow G \ (X \ x)$ $X \in Sols \ (\lambda t. \ f) \ T \ S \ 0 \ s$
and *main*: $s \in S \longrightarrow (\forall t \in T. \ (\forall \tau. \ \tau \in T \land \tau \le t \longrightarrow G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$
have $s \in S$
using *ivp-solsD*[*OF hyps*(*3*)] *init-time* **by** *auto*
hence $\forall \tau \in down \ T \ t. \ X \ \tau = \varphi \ \tau \ s$
using *eq-solution hyps* **by** *blast*
thus $Q \ (X \ t)$
using *main* ‹$s \in S$› *hyps* **by** *fastforce*
qed

lemma *sH-g-ode*: *rel-kat.Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ 0)$ $\lceil Q \rceil$ =
$(\forall s \in S. \ P \ s \longrightarrow (\forall t \in T. \ (\forall \tau \in down \ T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
fix *s t*
assume *hyps*: $s \in S$ $P \ s$ $t \in T$ $\forall \tau. \ \tau \in T \land \tau \le t \longrightarrow G \ (\varphi \ \tau \ s)$
and *main*: $\forall s. \ P \ s \longrightarrow (\forall X \in Sols \ (\lambda t. \ f) \ T \ S \ 0 \ s. \ \forall t \in T. \ (\forall \tau. \ \tau \in T \land \tau \le t \longrightarrow G \ (X \ \tau)) \longrightarrow Q \ (X \ t))$
hence $(\lambda t. \ \varphi \ t \ s) \in Sols \ (\lambda t. \ f) \ T \ S \ 0 \ s$
using *in-ivp-sols* **by** *blast*
thus $Q \ (\varphi \ t \ s)$
using *main hyps* **by** *fastforce*
next
fix *s X t*
assume *hyps*: $P \ s$ $X \in Sols \ (\lambda t. \ f) \ T \ S \ 0 \ s$ $t \in T$ $\forall \tau. \ \tau \in T \land \tau \le t \longrightarrow G \ (X \ \tau)$
and *main*: $\forall s \in S. \ P \ s \longrightarrow (\forall t \in T. \ (\forall \tau. \ \tau \in T \land \tau \le t \longrightarrow G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$
hence *obs*: $s \in S$
using *ivp-sols-def*[*of* $\lambda t. \ f$] *init-time* **by** *auto*
hence $\forall \tau \in down \ T \ t. \ X \ \tau = \varphi \ \tau \ s$
using *eq-solution hyps* **by** *blast*
thus $Q \ (X \ t)$
using *hyps main obs* **by** *auto*

qed

lemma *sH-g-ode-ivl*: $\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ $(x' = f$ & G *on* $\{0..\tau\}$ S @ $0)$ $\lceil Q \rceil$ =
$(\forall s \in S.\ P\ s \longrightarrow (\forall t \in \{0..\tau\}.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
  fix $s\ t$
  assume *hyps*: $0 \leq \tau$ $\tau \in T$ $s \in S$ $P\ s$ $t \in \{0..\tau\}$ $\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)$
    and *main*: $\forall s.\ P\ s \longrightarrow (\forall X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s.\ \forall t \in \{0..\tau\}.$
$(\forall \tau'.\ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow G\ (X\ \tau')) \longrightarrow Q\ (X\ t))$
  hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$
    using *in-ivp-sols-ivl closed-segment-eq-real-ivl*[*of* $0\ \tau$] **by** *force*
  thus $Q\ (\varphi\ t\ s)$
    using *main hyps* **by** *fastforce*
next
  fix $s\ X\ t$
  assume *hyps*: $0 \leq \tau$ $\tau \in T$ $P\ s$ $X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$ $t \in \{0..\tau\}$
    $\forall \tau'.\ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow G\ (X\ \tau')$
    and *main*: $\forall s \in S.\ P\ s \longrightarrow (\forall t \in \{0..\tau\}.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  hence $s \in S$
    using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
  have *obs1*: $\forall \tau \in down\ \{0..\tau\}\ t.\ D\ X = (\lambda t.\ f\ (X\ t))$ *on* $\{0--\tau\}$
    apply(*clarsimp*, *rule has-vderiv-on-subset*)
    using *ivp-solsD*(*1*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  have *obs2*: $X\ 0 = s$ $\forall \tau \in down\ \{0..\tau\}\ t.\ X \in \{0--\tau\} \to S$
    using *ivp-solsD*(*2*,*3*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  have $\forall \tau \in down\ \{0..\tau\}\ t.\ \tau \in T$
  using *subintervalI*[*OF init-time* ‹$\tau \in T$›] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  hence $\forall \tau \in down\ \{0..\tau\}\ t.\ X\ \tau = \varphi\ \tau\ s$
    using *obs1 obs2* **apply**(*clarsimp*)
    by (*rule eq-solution-ivl*) (*auto simp*: *closed-segment-eq-real-ivl*)
  thus $Q\ (X\ t)$
    using *hyps main* ‹$s \in S$› **by** *auto*
qed

lemma *sH-orbit*:
  rel-kat.Hoare $\lceil P \rceil$ $(\{(s,s') \mid s\ s'.\ s' \in \gamma^{\varphi}\ s\})$ $\lceil Q \rceil$ = $(\forall s \in S.\ P\ s \longrightarrow (\forall\ t \in T.$
$Q\ (\varphi\ t\ s)))$
  using *sH-g-ode* **unfolding** *orbit-def g-ode-def* **by** *auto*

end

— Verification with differential invariants

definition *g-ode-inv* :: $(('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow$
  $real \Rightarrow 'a\ pred \Rightarrow 'a\ rel$ ((1x´=- & - on - - @ - DINV -))
  where $(x' = f$ & G *on* $T\ S$ @ t_0 *DINV* $I) = (x' = f$ & G *on* $T\ S$ @ $t_0)$

lemma *sH-g-orbital-guard*:

assumes $R = (\lambda s.\ G\ s \wedge Q\ s)$
shows *rel-kat.Hoare* $\lceil P \rceil$ ($x' = f$ & G *on* $T\ S$ @ t_0) $\lceil Q \rceil$ = *rel-kat.Hoare* $\lceil P \rceil$ ($x' = f$ & G *on* $T\ S$ @ t_0) $\lceil R \rceil$
using *assms* **unfolding** *g-orbital-eq rel-kat-H ivp-sols-def g-ode-def* **by** *auto*

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

lemma *sH-diff-inv*[*simp*]: *rel-kat.Hoare* $\lceil I \rceil$ ($x' = f$ & G *on* $T\ S$ @ t_0) $\lceil I \rceil$ = *diff-invariant* $I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq rel-kat-H g-orbital-eq g-ode-def* **by** *auto*

lemma *H-g-ode-inv*: *rel-kat.Hoare* $\lceil I \rceil$ ($x' = f$ & G *on* $T\ S$ @ t_0) $\lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow$
$\lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ ($x' = f$ & G *on* $T\ S$ @ t_0 *DINV* I) $\lceil Q \rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* q'=$\lceil \lambda s.\ I\ s \wedge G\ s \rceil$ **in** *rel-kat.H-consr*, *simp*)
apply(*subst sH-g-orbital-guard*[*symmetric*], *force*)
by (*rule-tac* I=I **in** *sH-g-orbital-inv*, *simp-all*)

0.12.3 Refinement Components

— Skip

lemma *R-skip*: ($\forall s.\ P\ s \longrightarrow Q\ s$) $\Longrightarrow$ *Id* $\leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
by (*simp add*: *rel-rkat.R2 rel-kat-H*)

— Composition

lemma *R-seq*: (*rel-R* $\lceil P \rceil$ $\lceil R \rceil$) ; (*rel-R* $\lceil R \rceil$ $\lceil Q \rceil$) $\leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
using *rel-rkat.R-seq* **by** *blast*

lemma *R-seq-rule*: $X \leq$ *rel-R* $\lceil P \rceil$ $\lceil R \rceil \Longrightarrow Y \leq$ *rel-R* $\lceil R \rceil$ $\lceil Q \rceil \Longrightarrow X;\ Y \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-seq*)

lemmas *R-seq-mono* = *relcomp-mono*

— Assignment

lemma *R-assign*: (x ::= e) $\leq$ *rel-R* $\lceil \lambda s.\ P\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j) \rceil$ $\lceil P \rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-assign*, *clarsimp simp*: *fun-upd-def*)

lemma *R-assign-rule*: $(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ ((((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \le$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-assign[symmetric]* **by** (*rule rel-rkat.R2*)

lemma *R-assignl*: $P = (\lambda s.\ R\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j)) \Longrightarrow (x ::= e)$; *rel-R* $\lceil R\rceil\ \lceil Q\rceil \le$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-assign-rule*, *simp-all*)

lemma *R-assignr*: $R = (\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j)) \Longrightarrow$ *rel-R* $\lceil P\rceil\ \lceil R\rceil$; $(x ::= e) \le$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-assign-rule*, *simp*)

lemma $(x ::= e)$; *rel-R* $\lceil Q\rceil\ \lceil Q\rceil \le$ *rel-R* $\lceil(\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j))\rceil\ \lceil Q\rceil$
by (*rule R-assignl*) *simp*

lemma *rel-R* $\lceil Q\rceil\ \lceil(\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j))\rceil$; $(x ::= e) \le$ *rel-R* $\lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-assignr*) *simp*

— Conditional

lemma *R-cond*: (*IF B THEN rel-R* $\lceil\lambda s.\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil$ *ELSE rel-R* $\lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil) \le$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
using *rel-rkat.R-cond*[*of* $\lceil B\rceil\ \lceil P\rceil\ \lceil Q\rceil$] **by** *simp*

lemma *R-cond-mono*: $X \le X' \Longrightarrow Y \le Y' \Longrightarrow$ (*IF P THEN X ELSE Y*) $\le$ *IF P THEN X′ ELSE Y′*
by (*auto simp*: *rel-kat.kat-cond-def*)

— While loop

lemma *R-while*: *WHILE Q INV I DO* (*rel-R* $\lceil\lambda s.\ P\ s \wedge Q\ s\rceil\ \lceil P\rceil) \le$ *rel-R* $\lceil P\rceil\ \lceil\lambda s.\ P\ s \wedge \neg\ Q\ s\rceil$
unfolding *rel-kat.kat-while-inv-def* **using** *rel-rkat.R-while*[*of* $\lceil Q\rceil\ \lceil P\rceil$] **by** *simp*

lemma *R-while-mono*: $X \le X' \Longrightarrow$ (*WHILE P INV I DO X*) $\subseteq$ *WHILE P INV I DO X′*
by (*simp add*: *rel-kat.kat-while-inv-def rel-kat.kat-while-def rel-uq.mult-isol rel-uq.mult-isor rel-ka.star-iso*)

— Finite loop

lemma *R-loop*: $X \le$ *rel-R* $\lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \le \lceil I\rceil \Longrightarrow \lceil I\rceil \le \lceil Q\rceil \Longrightarrow$ *LOOP X INV I* $\le$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *rel-rkat.spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \leq X' \Longrightarrow LOOP\ X\ INV\ I \subseteq LOOP\ X'\ INV\ I$
unfolding *rel-kat.kat-loop-inv-def* **by** (*simp add*: *rel-ka.star-iso*)

— Evolution command (flow)

lemma *R-g-evol*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $(EVOL\ \varphi\ G\ T) \leq rel\text{-}R\ \lceil\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)\rceil\ \lceil P\rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-evol*, *simp*)

lemma *R-g-evol-rule*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $(\forall s.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (EVOL\ \varphi\ G\ T) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-evol*[*symmetric*] *rel-rkat.spec-def* **.**

lemma *R-g-evoll*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $P = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(EVOL\ \varphi\ G\ T)\ ;\ rel\text{-}R\ \lceil R\rceil\ \lceil Q\rceil \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R*=*R* **in** *R-seq-rule*)
by (*rule-tac R-g-evol-rule*, *simp-all*)

lemma *R-g-evolr*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $R = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$rel\text{-}R\ \lceil P\rceil\ \lceil R\rceil;\ (EVOL\ \varphi\ G\ T) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R*=*R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-evol-rule*, *simp*)

lemma
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $EVOL\ \varphi\ G\ T\ ;\ rel\text{-}R\ \lceil Q\rceil\ \lceil Q\rceil \leq rel\text{-}R\ \lceil\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
by (*rule R-g-evoll*) *simp*

lemma
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
shows $rel\text{-}R\ \lceil Q\rceil\ \lceil\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil;\ EVOL\ \varphi\ G\ T \leq rel\text{-}R\ \lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-g-evolr*) *simp*

— Evolution command (ode)

context *local-flow*
begin

lemma *R-g-ode*: $(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq rel\text{-}R\ \lceil\lambda s.\ s{\in}S \longrightarrow (\forall t{\in}T.$

$(\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-ode*, *simp*)

lemma *R-g-ode-rule*: $(\forall s \in S.\ P\ s \longrightarrow (\forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode*[*symmetric*] **by** (*rule rel-rkat.R2*)

lemma *R-g-odel*: $P = (\lambda s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x' = f\ \&\ G\ on\ T\ S\ @\ 0)\ ;\ rel\text{-}R\ \lceil R\rceil\ \lceil Q\rceil \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-g-ode-rule*, *simp-all*)

lemma *R-g-oder*: $R = (\lambda s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$

$rel\text{-}R\ \lceil P\rceil\ \lceil R\rceil;\ (x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-ode-rule*, *simp*)

lemma $(x' = f\ \&\ G\ on\ T\ S\ @\ 0)\ ;\ rel\text{-}R\ \lceil Q\rceil\ \lceil Q\rceil \leq rel\text{-}R\ \lceil \lambda s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
by (*rule R-g-odel*) *simp*

lemma $rel\text{-}R\ \lceil Q\rceil\ \lceil \lambda s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil;\ (x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq rel\text{-}R\ \lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-g-oder*) *simp*

lemma *R-g-ode-ivl*:
$\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow (\forall s \in S.\ P\ s \longrightarrow (\forall t \in \{0..\tau\}.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = f\ \&\ G\ on\ \{0..\tau\}\ S\ @\ 0) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode-ivl*[*symmetric*] **by** (*rule rel-rkat.R2*)

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $I\ f\ T\ S\ t_0\ G \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$(x'=f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) \leq rel\text{-}R\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *rel-rkat.spec-def* **by** (*auto simp*: *H-g-ode-inv*)

0.12.4 Derivation of the rules of dL

We derive a generalised version of some domain specific rules of differential dynamic logic (dL).

lemma *diff-solve-axiom*:
fixes *c*::'*a*::{*heine-borel*, *banach*}
assumes $0 \in T$ **and** *is-interval T open T*

and $\forall s.\ P\ s \longrightarrow (\forall t{\in}T.\ (\mathcal{P}\ (\lambda\ t.\ s + t *_R c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (s + t *_R c))$
shows *rel-kat.Hoare* $\lceil P\rceil$ $(x'{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)$ $\lceil Q\rceil$
apply(*subst local-flow.sH-g-ode*[**where** $f{=}\lambda s.\ c$ **and** $\varphi{=}(\lambda\ t\ x.\ x + t *_R c)$])
using *line-is-local-flow assms* **by** *auto*

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

lemma *diff-weak-rule*:
assumes $\lceil G\rceil \leq \lceil Q\rceil$
shows *rel-kat.Hoare* $\lceil P\rceil$ $(x'{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q\rceil$
using *assms* **unfolding** *g-orbital-eq rel-kat-H ivp-sols-def g-ode-def* **by** *auto*

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

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

abbreviation *g-global-ode-inv* :: ((*'a*::*banach*)⇒*'a*) ⇒ *'a pred* ⇒ *'a pred* ⇒ *'a rel*
((*1x´*=- & - *DINV* -)) **where** $(x'{=}\ f\ \&\ G\ DINV\ I) \equiv (x'{=}\ f\ \&\ G\ on\ UNIV\ UNIV\ @\ 0\ DINV\ I)$

end

0.12.5 Examples

We prove partial correctness specifications of some hybrid systems with our refinement and verification components.

theory *kat2rel-examples*
imports *kat2rel*

begin

Pendulum

The ODEs $x'\ t = y\ t$ and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use *s$1* to represent the x-coordinate and *s$2* for the y-coordinate. We prove that this motion remains circular.

abbreviation *fpend* :: *real^2* ⇒ *real^2* (*f*)
where *f s* ≡ (χ *i*. *if i=1 then s$2 else* −*s$1*)

abbreviation *pend-flow* :: *real* ⇒ *real^2* ⇒ *real^2* (φ)
where φ τ *s* ≡ (χ *i*. *if i* = *1 then s$1* · *cos* τ + *s$2* · *sin* τ
else − *s$1* · *sin* τ + *s$2* · *cos* τ)

— Verified with annotated dynamics

lemma *pendulum-dyn*: *rel-kat.Hoare* $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (*EVOL* φ *G T*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by *simp*

— Verified with differential invariants

lemma *pendulum-inv*: *rel-kat.Hoare*
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (*x´=f* & *G*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified with the flow

lemma *local-flow-pend*: *local-flow f UNIV UNIV* φ
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *pendulum-flow*: *rel-kat.Hoare*
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (*x´=f* & *G*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*simp only*: *local-flow.sH-g-ode*[*OF local-flow-pend*], *simp*)

no-notation *fpend* (*f*)

and *pend-flow* (φ)

Bouncing Ball

A ball is dropped from rest at an initial height h. The motion is described with the free-fall equations $x'\ t = v\ t$ and $v'\ t = g$ where g is the constant acceleration due to gravity. The bounce is modelled with a variable assigntment that flips the velocity, thus it is a completely elastic collision with the ground. We use $s\$1$ to ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

abbreviation *fball* :: *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (f)
where $f\ g\ s \equiv (\chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ g)$

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real*ˆ*2* ⇒ *real*ˆ*2* (φ)
where $\varphi\ g\ \tau\ s \equiv (\chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ g \cdot \tau\ \hat{}\ 2/2 + s\$2 \cdot \tau + s\$1\ \mathit{else}\ g \cdot \tau + s\$2)$

— Verified with differential invariants

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

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

lemma *fball-invariant*:
fixes *g h* :: *real*
defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows *diff-invariant I* (f g) *UNIV UNIV 0 G*
unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
by(*auto intro*!: *poly-derivatives*)

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *rel-kat.Hoare*
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$
(*LOOP*

((*x´= f g* & (λ *s. s$1* $\geq$ *0*) *DINV* (λ*s. 2* $\cdot$ *g* $\cdot$ *s$1* $-$ *2* $\cdot$ *g* $\cdot$ *h* $-$ *s$2* $\cdot$ *s$2* = *0*));
(*IF* (λ *s. s$1* = *0*) *THEN* (*2* ::= (λ*s.* $-$ *s$2*)) *ELSE skip*))
INV (λ*s. 0* $\leq$ *s$1* $\wedge$ *2* $\cdot$ *g* $\cdot$ *s$1* = *2* $\cdot$ *g* $\cdot$ *h* + *s$2* $\cdot$ *s$2*)
) ⌈λ*s. 0* $\leq$ *s$1* $\wedge$ *s$1* $\leq$ *h*⌉
apply(*rule H-loopI*)
apply(*rule H-seq*[**where** *R*=λ*s. 0* $\leq$ *s$1* $\wedge$ *2* $\cdot$ *g* $\cdot$ *s$1* = *2* $\cdot$ *g* $\cdot$ *h* + *s$2* $\cdot$ *s$2*])
apply(*rule H-g-ode-inv*)
by (*auto simp*: *bb-real-arith intro*!: *poly-derivatives diff-invariant-rules*)

— Verified with annotated dynamics

lemma [*bb-real-arith*]:
assumes *invar*: *2* $\cdot$ *g* $\cdot$ *x* = *2* $\cdot$ *g* $\cdot$ *h* + *v* $\cdot$ *v*
and *pos*: *g* $\cdot$ τ^2 / *2* + *v* $\cdot$ τ + (*x*::*real*) = *0*
shows *2* $\cdot$ *g* $\cdot$ *h* + ($-$ (*g* $\cdot$ τ) $-$ *v*) $\cdot$ ($-$ (*g* $\cdot$ τ) $-$ *v*) = *0*
and *2* $\cdot$ *g* $\cdot$ *h* + (*g* $\cdot$ τ $\cdot$ (*g* $\cdot$ τ + *v*) + *v* $\cdot$ (*g* $\cdot$ τ + *v*)) = *0*
proof−
from *pos* **have** *g* $\cdot$ τ^2 + *2* $\cdot$ *v* $\cdot$ τ + *2* $\cdot$ *x* = *0* **by** *auto*
then have g^2 $\cdot$ τ^2 + *2* $\cdot$ *g* $\cdot$ *v* $\cdot$ τ + *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 $\cdot$ τ^2 + *2* $\cdot$ *g* $\cdot$ *v* $\cdot$ τ + v^2 + *2* $\cdot$ *g* $\cdot$ *h* = *0*
using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
hence *obs*: (*g* $\cdot$ τ + *v*)^2 + *2* $\cdot$ *g* $\cdot$ *h* = *0*
apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
thus *2* $\cdot$ *g* $\cdot$ *h* + (*g* $\cdot$ τ $\cdot$ (*g* $\cdot$ τ + *v*) + *v* $\cdot$ (*g* $\cdot$ τ + *v*)) = *0*
by (*simp add*: *monoid-mult-class.power2-eq-square*)
have *2* $\cdot$ *g* $\cdot$ *h* + ($-$ ((*g* $\cdot$ τ) + *v*))^2 = *0*
using *obs* **by** (*metis Groups.add-ac*(*2*) *power2-minus*)
thus *2* $\cdot$ *g* $\cdot$ *h* + ($-$ (*g* $\cdot$ τ) $-$ *v*) $\cdot$ ($-$ (*g* $\cdot$ τ) $-$ *v*) = *0*
by (*simp add*: *monoid-mult-class.power2-eq-square*)
qed

lemma [*bb-real-arith*]:
assumes *invar*: *2* $\cdot$ *g* $\cdot$ *x* = *2* $\cdot$ *g* $\cdot$ *h* + *v* $\cdot$ *v*
shows *2* $\cdot$ *g* $\cdot$ (*g* $\cdot$ τ^2 / *2* + *v* $\cdot$ τ + (*x*::*real*)) =
2 $\cdot$ *g* $\cdot$ *h* + (*g* $\cdot$ τ $\cdot$ (*g* $\cdot$ τ + *v*) + *v* $\cdot$ (*g* $\cdot$ τ + *v*)) (**is** *?lhs* = *?rhs*)
proof−
have *?lhs* = g^2 $\cdot$ τ^2 + *2* $\cdot$ *g* $\cdot$ *v* $\cdot$ τ + *2* $\cdot$ *g* $\cdot$ *x*
by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
also have ... = g^2 $\cdot$ τ^2 + *2* $\cdot$ *g* $\cdot$ *v* $\cdot$ τ + *2* $\cdot$ *g* $\cdot$ *h* + *v* $\cdot$ *v* (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** *?rhs* = *g* $\cdot$ *g* $\cdot$ (τ $\cdot$ τ) + *2* $\cdot$ *g* $\cdot$ *v* $\cdot$ τ + *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-dyn*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *rel-kat.Hoare*
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$
(*LOOP*
((*EVOL* (φ *g*) (λ *s*. $s\$1 \geq 0$) *T*);
(*IF* (λ *s*. $s\$1 = 0$) *THEN* (*2* ::= ($\lambda s.\ - s\$2$)) *ELSE skip*))
INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
apply(*rule H-loopI*, *rule H-seq*[**where** $R{=}\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$])
by (*auto simp*: *bb-real-arith*)

— Verified with the flow

lemma *local-flow-ball*: *local-flow* (*f g*) *UNIV UNIV* (φ *g*)
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *rel-kat.Hoare*
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$
(*LOOP*
(($x' = f\ g$ & (λ *s*. $s\$1 \geq 0$));
(*IF* (λ *s*. $s\$1 = 0$) *THEN* (*2* ::= ($\lambda s.\ - s\$2$)) *ELSE skip*))
INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
apply(*rule H-loopI*)
apply(*rule H-seq*[**where** $R{=}\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$])
apply(*subst local-flow.sH-g-ode*[*OF local-flow-ball*])
apply(*force simp*: *bb-real-arith*)
by (*rule H-cond*) (*auto simp*: *bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g < (0{::}real) \Longrightarrow 0 \leq h \Longrightarrow$
2 ::= ($\lambda s.\ - s\$2$) $\leq$ *rel-R*
$\lceil \lambda s.\ s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
by (*rule R-assign-rule*, *auto*)

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows *rel-R* $\lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil\ \lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil \geq$
(*LOOP*
((*EVOL* (φ *g*) (λ *s*. $s\$1 \geq 0$) *T*);
(*IF* (λ *s*. $s\$1 = 0$) *THEN* ($2 ::= (\lambda s.\ - s\$2)$) *ELSE skip*))
INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$))
apply(*rule order-trans*)
apply(*rule R-loop-mono*) **defer**
apply(*rule R-loop*)
apply(*rule R-seq*)
using *assms* **apply**(*simp-all*, *force simp*: *bb-real-arith*)
apply(*rule R-seq-mono*) **defer**
apply(*rule order-trans*)
apply(*rule R-cond-mono*) **defer defer**
apply(*rule R-cond*) **defer**
using *R-bb-assign* **apply** *force*
apply(*rule R-skip*, *clarsimp*)
by (*rule R-g-evol-rule*, *force simp*: *bb-real-arith*)

no-notation *fball* (f)
and *ball-flow* (φ)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every τ minutes, it sets its chronometer to *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = - a * (T - U)$ where $U = L \geq 0$ when the heater is on, and $U = 0$ when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, and *3* is a variable to save temperature measurements. Finally, *4* states whether the heater is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *therm-vec-field* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*ˆ*4* $\Rightarrow$ *real*ˆ*4* (f)
where $f\ a\ L\ s \equiv (\chi\ i.\ \textit{if}\ i = 2\ \textit{then}\ 1\ \textit{else}\ (\textit{if}\ i = 1\ \textit{then}\ - a * (s\$1 - L)\ \textit{else}\ 0))$

abbreviation *therm-guard* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real*ˆ*4* $\Rightarrow$ *bool* (G)
where $G\ Tmin\ Tmax\ a\ L\ s \equiv (s\$2 \leq - (\ln\ ((L-(\textit{if}\ L=0\ \textit{then}\ Tmin\ \textit{else}\ Tmax))/(L-s\$3)))/a)$

abbreviation *therm-loop-inv* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*ˆ*4* $\Rightarrow$ *bool* (I)
where $I\ Tmin\ Tmax\ s \equiv Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *therm-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real*ˆ*4* $\Rightarrow$ *real*ˆ*4* (φ)
where $\varphi\ a\ L\ \tau\ s \equiv (\chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ - \exp(-a * \tau) * (L - s\$1) + L\ \textit{else}$
$(\textit{if}\ i = 2\ \textit{then}\ \tau + s\$2\ \textit{else}\ s\$i))$

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \Longrightarrow \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$
proof(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)
  assume *a1*: $0 < a$
  have *f2*: $\bigwedge r\ ra.\ |(r::real) + - ra| = |ra + - r|$
    by (*metis abs-minus-commute minus-real-def*)
  have $\bigwedge r\ ra\ rb.\ (r::real) * ra + - (r * rb) = r * (ra + - rb)$
    by (*metis minus-real-def right-diff-distrib*)
  hence $|a * (s_1\$1 + - L) + - (a * (s_2\$1 + - L))| = a * |s_1\$1 + - s_2\$1|$
    using *a1* **by** (*simp add*: *abs-mult*)
  thus $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$
    using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-therm-dyn*:
  assumes $0 < (a::real)$
  shows *local-lipschitz UNIV UNIV* $(\lambda t::real.\ f\ a\ L)$
  apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
  apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*)
  using *assms* **apply**(*simp-all add*: *norm-diff-therm-dyn*)
  apply(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *clarsimp*)
  unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-therm*: $a > 0 \Longrightarrow$ *local-flow* $(f\ a\ L)$ *UNIV UNIV* $(\varphi\ a\ L)$
  by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-therm-dyn*
      simp: *forall-4 vec-eq-iff*)

lemma *therm-dyn-down-real-arith*:
  assumes $a > 0$ **and** *Thyps*: $0 < Tmin\ Tmin \leq T\ T \leq Tmax$
    and *thyps*: $0 \leq (\tau::real)\ \forall\tau\in\{0..\tau\}.\ \tau \leq - (ln\ (Tmin\ /\ T)\ /\ a)$
  shows $Tmin \leq exp\ (-a * \tau) * T$ **and** $exp\ (-a * \tau) * T \leq Tmax$
proof−
  have $0 \leq \tau \wedge \tau \leq - (ln\ (Tmin\ /\ T)\ /\ a)$
    using *thyps* **by** *auto*
  hence $ln\ (Tmin\ /\ T) \leq - a * \tau \wedge - a * \tau \leq 0$
    using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
  also have $Tmin\ /\ T > 0$
    using *Thyps* **by** *auto*
  ultimately have *obs*: $Tmin\ /\ T \leq exp\ (-a * \tau)\ exp\ (-a * \tau) \leq 1$
    using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
  thus $Tmin \leq exp\ (-a * \tau) * T$
    using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
  show $exp\ (-a * \tau) * T \leq Tmax$
    using *Thyps mult-left-le-one-le*[*OF - exp-ge-zero obs*(*2*), *of T*]
      less-eq-real-def order-trans-rules(*23*) **by** *blast*
qed

lemma *therm-dyn-up-real-arith*:
  assumes $a > 0$ **and** *Thyps*: $Tmin \leq T$ $T \leq Tmax$ $Tmax < (L::real)$
    and *thyps*: $0 \leq \tau$ $\forall \tau \in \{0..\tau\}.\ \tau \leq -\ (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
  shows $L - Tmax \leq exp\ (-(a * \tau)) * (L - T)$
    and $L - exp\ (-(a * \tau)) * (L - T) \leq Tmax$
    and $Tmin \leq L - exp\ (-(a * \tau)) * (L - T)$
proof−
  have $0 \leq \tau \wedge \tau \leq -\ (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
    using *thyps* **by** *auto*
  hence $ln\ ((L - Tmax)\ /\ (L - T)) \leq -\ a * \tau \wedge -\ a * \tau \leq 0$
    using *assms(1) divide-le-cancel* **by** *fastforce*
  also have $(L - Tmax)\ /\ (L - T) > 0$
    using *Thyps* **by** *auto*
  ultimately have $(L - Tmax)\ /\ (L - T) \leq exp\ (-a * \tau) \wedge exp\ (-a * \tau) \leq 1$
    using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
  moreover have $L - T > 0$
    using *Thyps* **by** *auto*
  ultimately have *obs*: $(L - Tmax) \leq exp\ (-a * \tau) * (L - T) \wedge exp\ (-a * \tau)$
$* (L - T) \leq (L - T)$
    by (*simp add*: *pos-divide-le-eq*)
  thus $(L - Tmax) \leq exp\ (-(a * \tau)) * (L - T)$
    by *auto*
  thus $L - exp\ (-(a * \tau)) * (L - T) \leq Tmax$
    by *auto*
  show $Tmin \leq L - exp\ (-(a * \tau)) * (L - T)$
    using *Thyps* **and** *obs* **by** *auto*
qed

lemmas *H-g-ode-therm* = *local-flow.sH-g-ode-ivl*[*OF local-flow-therm - UNIV-I*]

lemma *thermostat-flow*:
  assumes $0 < a$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
  shows *rel-kat.Hoare* ⌈*I Tmin Tmax*⌉
  (*LOOP* (
    — control
    (2 ::= ($\lambda s.\ 0$));
    (3 ::= ($\lambda s.\ s\$1$));
    (*IF* ($\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1$) *THEN*
      (4 ::= ($\lambda s.1$))
     ELSE IF ($\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1$) *THEN*
      (4 ::= ($\lambda s.0$))
     ELSE skip);
    — dynamics
    (*IF* ($\lambda s.\ s\$4 = 0$) *THEN*
      (x'= *f a 0* & *G Tmin Tmax a 0 on* $\{0..\tau\}$ *UNIV* @ *0*)
    ELSE
      (x'= *f a L* & *G Tmin Tmax a L on* $\{0..\tau\}$ *UNIV* @ *0*))
) *INV I Tmin Tmax*)

$\lceil I\ Tmin\ Tmax \rceil$
apply(*rule H-loopI*)
apply(*rule-tac R*=λs. *I Tmin Tmax s* $\wedge$ *s*\$*2*=*0* $\wedge$ *s*\$*3* = *s*\$*1* **in** *H-seq*)
apply(*rule-tac R*=λs. *I Tmin Tmax s*$\wedge$ *s*\$*2*=*0* $\wedge$ *s*\$*3* = *s*\$*1* **in** *H-seq*)
apply(*rule-tac R*=λs. *I Tmin Tmax s* $\wedge$ *s*\$*2*=*0* **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp-all add*: *H-g-ode-therm*[*OF assms*(*1*,*2*)])+
using *therm-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *therm-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of* - *Tmax*] **by** *auto*

— Refined with the flow

lemma *R-therm-dyn-down*:
assumes $a > 0$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *rel-R* $\lceil \lambda s.\ s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil \ge$
($x\acute{}= f\ a\ 0\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0$)
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *assms therm-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of* - *Tmax*] **by** *auto*

lemma *R-therm-dyn-up*:
assumes $a > 0$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *rel-R* $\lceil \lambda s.\ s\$4 \neq 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil \ge$
($x\acute{}= f\ a\ L\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ \{0..\tau\}\ UNIV\ @\ 0$)
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *assms therm-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*] **by** *auto*

lemma *R-therm-dyn*:
assumes $a > 0$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *rel-R* $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil \ge$
(*IF* ($\lambda s.\ s\$4 = 0$) *THEN*
($x\acute{}= f\ a\ 0\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0$)
ELSE
($x\acute{}= f\ a\ L\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ \{0..\tau\}\ UNIV\ @\ 0$))
apply(*rule order-trans*, *rule R-cond-mono*)
using *R-therm-dyn-down*[*OF assms*] *R-therm-dyn-up*[*OF assms*] **by** (*auto intro*!: *R-cond*)

lemma *R-therm-assign1*: *rel-R* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \rceil$ $\ge$ ($2 ::= (\lambda s.\ 0)$)
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-assign2*:
rel-R $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \rceil$ $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil \ge (3 ::= (\lambda s.\ s\$1))$
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-ctrl*:

```
  rel-R ⌈I Tmin Tmax⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ≥
  (2 ::= (λs. 0));
  (3 ::= (λs. s$1));
  (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
    (4 ::= (λs.1))
   ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax − 1) THEN
    (4 ::= (λs.0))
   ELSE skip)
  apply(rule R-seq-rule)+
    apply(rule R-therm-assign1)
   apply(rule R-therm-assign2)
  apply(rule order-trans)
   apply(rule R-cond-mono)
    apply(rule R-assign-rule) defer
    apply(rule R-cond-mono)
     apply(rule R-assign-rule) defer
     apply(rule R-skip) defer
     apply(rule order-trans)
      apply(rule R-cond-mono)
       apply force
  by (rule R-cond)+ auto

lemma R-therm-loop: rel-R ⌈I Tmin Tmax⌉ ⌈I Tmin Tmax⌉ ≥
  (LOOP
    rel-R ⌈I Tmin Tmax⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉;
    rel-R ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉
  INV I Tmin Tmax)
  by (intro R-loop R-seq, simp-all)

lemma R-thermostat-flow:
  assumes a > 0 and 0 ≤ τ and 0 < Tmin and Tmax < L
  shows rel-R ⌈I Tmin Tmax⌉ ⌈I Tmin Tmax⌉ ≥
  (LOOP (
    — control
    (2 ::= (λs. 0));(3 ::= (λs. s$1));
    (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
      (4 ::= (λs.1))
     ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax − 1) THEN
      (4 ::= (λs.0))
     ELSE skip);
    — dynamics
    (IF (λs. s$4 = 0) THEN
      (x´= f a 0 & G Tmin Tmax a 0 on {0..τ} UNIV @ 0)
    ELSE
      (x´= f a L & G Tmin Tmax a L on {0..τ} UNIV @ 0))
  ) INV I Tmin Tmax)
  by (intro order-trans[OF - R-therm-loop] R-loop-mono
      R-seq-mono R-therm-ctrl R-therm-dyn[OF assms])
```

no-notation *therm-vec-field* (f)
  and *therm-flow* (φ)
  and *therm-guard* (G)
  and *therm-loop-inv* (I)

Water tank

— Variation of Hespanha and [**?**]

abbreviation *tank-vec-field* :: $real \Rightarrow real\hat{\ }4 \Rightarrow real\hat{\ }4$ (f)
where $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

abbreviation *tank-flow* :: $real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow real\hat{\ }4$ (φ)
where $\varphi\ k\ \tau\ s \equiv (\chi\ i.\ if\ i = 1\ then\ k * \tau + s\$1\ else$
$(if\ i = 2\ then\ \tau + s\$2\ else\ s\$i))$

abbreviation *tank-guard* :: $real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (G)
where $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: $real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (I)
where $I\ hmin\ hmax\ s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *tank-diff-inv* :: $real \Rightarrow real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (dI)
where $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$
$hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 =0 \vee s\$4 = 1)$

— Verified with the flow

lemma *local-flow-tank*: *local-flow* ($f\ k$) *UNIV UNIV* ($\varphi\ k$)
apply (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def*, *unfold UNIV-4*)
by (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff*)

lemma *tank-arith*:
assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau \in \{0..\tau\}.\ \tau \leq - ((hmin - y) / c_o) \Longrightarrow hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0..\tau\}.\ \tau \leq (hmax - y) / (c_i - c_o) \Longrightarrow (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) \cdot \tau + y$
and $y \leq hmax \Longrightarrow y - c_o \cdot \tau \leq hmax$
apply(*simp-all add*: *field-simps le-divide-eq assms*)
using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

lemmas *H-g-ode-tank* = *local-flow.sH-g-ode-ivl*[*OF local-flow-tank - UNIV-I*]

lemma *tank-flow*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$

shows *rel-kat.Hoare* ⌈*I hmin hmax*⌉
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0* ∧ *s$3* ≤ *hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1* ∧ *s$3* ≥ *hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s. s$4 = 0*) *THEN* (*x´= f* ($c_i - c_o$) & *G hmax* ($c_i - c_o$) *on* {*0..τ*} *UNIV* @ *0*)
ELSE (*x´= f* ($-c_o$) & *G hmin* ($-c_o$) *on* {*0..τ*} *UNIV* @ *0*)))
INV I hmin hmax) ⌈*I hmin hmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R*=λ*s. I hmin hmax s* ∧ *s$2=0* ∧ *s$3 = s$1* **in** *H-seq*)
apply(*rule-tac R*=λ*s. I hmin hmax s* ∧ *s$2=0* ∧ *s$3 = s$1* **in** *H-seq*)
apply(*rule-tac R*=λ*s. I hmin hmax s* ∧ *s$2=0* **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp-all add*: *H-g-ode-tank*[*OF assms*(*1*)])
using *assms tank-arith*[*OF - assms*(*2*,*3*)] **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:
0 ≤ τ ⟹ *diff-invariant* (*dI hmin hmax k*) (*f k*) {*0..τ*} *UNIV 0 Guard*
apply(*intro diff-invariant-conj-rule*)
apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
apply(*rule-tac* ν′=λ*t. 0* **and** μ′=λ*t. 1* **in** *diff-invariant-leq-rule*, *simp-all*)
apply(*rule-tac* ν′=λ*t. 0* **and** μ′=λ*t. 0* **in** *diff-invariant-leq-rule*, *simp-all*)
apply(*force intro*!: *poly-derivatives*)+
by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

lemma *tank-inv-arith1*:
assumes $0 \le (\tau::real)$ **and** $c_o < c_i$ **and** *b*: $hmin \le y_0$ **and** *g*: $\tau \le (hmax - y_0) / (c_i - c_o)$
shows $hmin \le (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \le hmax$
proof−
have $(c_i - c_o) \cdot \tau \le (hmax - y_0)$
using *g assms*(*2*,*3*) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
thus $(c_i - c_o) \cdot \tau + y_0 \le hmax$
by *auto*
show $hmin \le (c_i - c_o) \cdot \tau + y_0$
using *b assms*(*1*,*2*) **by** (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)
qed

lemma *tank-inv-arith2*:
assumes $0 \le (\tau::real)$ **and** $0 < c_o$ **and** *b*: $y_0 \le hmax$ **and** *g*: $\tau \le -((hmin - y_0) / c_o)$
shows $hmin \le y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \le hmax$
proof−
have $\tau \cdot c_o \le y_0 - hmin$

```
    using g ‹0 < c_o› pos-le-minus-divide-eq by fastforce
  thus hmin ≤ y_0 − c_o · τ
    by (auto simp: mult.commute)
  show y_0 − c_o · τ ≤ hmax
    using b assms(1,2) by (smt mult-nonneg-nonneg)
qed

lemma tank-inv:
  assumes 0 ≤ τ and 0 < c_o and c_o < c_i
  shows rel-kat.Hoare ⌈I hmin hmax⌉
  (LOOP
    — control
    ((2 ::=(λs.0));(3 ::=(λs. s$1));
    (IF (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) THEN (4 ::= (λs.1)) ELSE
    (IF (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) THEN (4 ::= (λs.0)) ELSE skip));
    — dynamics
    (IF (λs. s$4 = 0) THEN
       (x´= f (c_i−c_o) & G hmax (c_i−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin
hmax (c_i−c_o)))
      ELSE
       (x´= f (−c_o) & G hmin (−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin hmax
(−c_o)))) )
  INV I hmin hmax) ⌈I hmin hmax⌉
  apply(rule H-loopI)
    apply(rule-tac R=λs. I hmin hmax s ∧ s$2=0 ∧ s$3 = s$1 in H-seq)
     apply(rule-tac R=λs. I hmin hmax s ∧ s$2=0 ∧ s$3 = s$1 in H-seq)
      apply(rule-tac R=λs. I hmin hmax s ∧ s$2=0 in H-seq, simp, simp)
     apply(rule H-cond, simp)
     apply(rule H-cond, simp, simp)
    apply(rule H-cond)
     apply(rule H-g-ode-inv)
  using assms tank-inv-arith1 apply(force simp: tank-diff-inv, simp, clarsimp)
    apply(rule H-g-ode-inv)
  using assms tank-diff-inv[of - −c_o hmin hmax] tank-inv-arith2 by auto

— Refined with differential invariants

lemma R-tank-inv:
  assumes 0 ≤ τ and 0 < c_o and c_o < c_i
  shows rel-R ⌈I hmin hmax⌉ ⌈I hmin hmax⌉ ≥
  (LOOP
    — control
    ((2 ::=(λs.0));(3 ::=(λs. s$1));
    (IF (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) THEN (4 ::= (λs.1)) ELSE
    (IF (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) THEN (4 ::= (λs.0)) ELSE skip));
    — dynamics
    (IF (λs. s$4 = 0) THEN
       (x´= f (c_i−c_o) & G hmax (c_i−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin
hmax (c_i−c_o)))
```

ELSE
(x´= f $(-c_o)$ *& G hmin* $(-c_o)$ *on* $\{0..\tau\}$ *UNIV @ 0 DINV (dI hmin hmax* $(-c_o)$*)))))*
INV I hmin hmax) (**is** *LOOP (?ctrl;?dyn) INV* - $\leq$ *?ref*)
proof–
— First we refine the control.
let *?Icntrl* = $\lambda s.$ *I hmin hmax s* $\wedge$ *s\$2 = 0* $\wedge$ *s\$3 = s\$1*
and *?cond* = $\lambda s.$ *s\$4 = 0* $\wedge$ *s\$3* $\leq$ *hmin + 1*
have *ifbranch1*: *4 ::=* $(\lambda s.1)$ $\leq$ *rel-R* $\lceil\lambda s.$ *?cond s* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*?Icntrl*$\rceil$ (**is** - $\leq$ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF* ($\lambda s.$ *s\$4 = 1* $\wedge$ *s\$3* $\geq$ *hmax* $-$ *1*) *THEN* (*4 ::=* $(\lambda s.0)$) *ELSE skip*) $\leq$
rel-R $\lceil\lambda s.$ $\neg$ *?cond s* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*?Icntrl*$\rceil$ (**is** - $\leq$ *?branch2*)
apply(*rule order-trans*, *rule R-cond-mono*) **defer defer**
by (*rule R-cond*) (*auto intro*!: *R-assign-rule R-skip*)
have *ifthenelse*: (*IF ?cond THEN ?branch1 ELSE ?branch2*) $\leq$ *rel-R* $\lceil$*?Icntrl*$\rceil$ $\lceil$*?Icntrl*$\rceil$ (**is** *?ifthenelse* $\leq$ -)
by (*rule R-cond*)
have (*IF ?cond THEN* (*4 ::=* $(\lambda s.1)$) *ELSE* (*IF* ($\lambda s.$ *s\$4 = 1* $\wedge$ *s\$3* $\geq$ *hmax* $-$ *1*) *THEN* (*4 ::=* $(\lambda s.0)$) *ELSE skip*)) $\leq$
rel-R $\lceil$*?Icntrl*$\rceil$ $\lceil$*?Icntrl*$\rceil$
apply(*rule-tac y=?ifthenelse* **in** *order-trans*, *rule R-cond-mono*)
using *ifbranch1 ifbranch2 ifthenelse* **by** *auto*
hence *ctrl*: *?ctrl* $\leq$ *rel-R* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$
apply(*rule-tac R=?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R=*$\lambda s.$ *I hmin hmax s* $\wedge$ *s\$2 = 0* **in** *R-seq-rule*)
by (*auto intro*!: *R-assign-rule*)
— Then we refine the dynamics.
have *dynup*: (*x´=f* (c_i-c_o) *& G hmax* (c_i-c_o) *on* $\{0..\tau\}$ *UNIV @ 0 DINV (dI hmin hmax* (c_i-c_o))) $\leq$
rel-R $\lceil\lambda s.$ *s\$4 = 0* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule R-g-ode-inv*[*OF tank-diff-inv*[*OF assms*(*1*)]])
using *assms* **by** (*auto simp*: *tank-inv-arith1*)
have *dyndown*: (*x´=f* $(-c_o)$ *& G hmin* $(-c_o)$ *on* $\{0..\tau\}$ *UNIV @ 0 DINV (dI hmin hmax* $(-c_o)$)) $\leq$
rel-R $\lceil\lambda s.$ *s\$4* $\neq$ *0* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule R-g-ode-inv*)
using *tank-diff-inv*[*OF assms*(*1*), *of* $-c_o$] *assms*
by (*auto simp*: *tank-inv-arith2*)
have *dyn*: *?dyn* $\leq$ *rel-R* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule order-trans*, *rule R-cond-mono*)
using *dynup dyndown* **by** (*auto intro*!: *R-cond*)
— Finally we put everything together.
have *pre-inv*: $\lceil$*I hmin hmax*$\rceil$ $\leq$ $\lceil$*I hmin hmax*$\rceil$
by *simp*
have *inv-pos*: $\lceil$*I hmin hmax*$\rceil$ $\leq$ $\lceil\lambda s.$ *hmin* $\leq$ *s\$1* $\wedge$ *s\$1* $\leq$ *hmax*$\rceil$
by *simp*
have *inv-inv*: *rel-R* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*rel-R* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$)

$\leq$ *rel-R* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
  by (*rule R-seq*)
 have *loopref*: *LOOP rel-R* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*rel-R* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$) *INV I hmin hmax* $\leq$ *?ref*
  apply(*rule R-loop*)
  using *pre-inv inv-inv inv-pos* **by** *auto*
 have *obs*: *?ctrl*;*?dyn* $\leq$ *rel-R* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*rel-R* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$)
  apply(*rule R-seq-mono*)
  using *ctrl dyn* **by** *auto*
 show *LOOP* (*?ctrl*;*?dyn*) *INV I hmin hmax* $\leq$ *?ref*
  by (*rule order-trans*[*OF - loopref*], *rule R-loop-mono*[*OF obs*])
qed

no-notation *tank-vec-field* (*f*)
  and *tank-flow* (φ)
  and *tank-guard* (*G*)
  and *tank-loop-inv* (*I*)
  and *tank-diff-inv* (*dI*)

end

0.13 Verification and refinement of HS in the relational KAT

We use our state transformers model to obtain verification and refinement components for hybrid programs. We devise three methods for reasoning with evolution commands and their continuous dynamics: providing flows, solutions or invariants.

theory *kat2ndfun*
 imports
  ../hs-prelims-ka
  ../hs-prelims-dyn-sys

begin

0.13.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
  and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)
  and *tau* (τ)
  and *Relation.relcomp* (**infixl** ; *75*)
  and *proto-near-quantale-class.bres* (**infixr** $\rightarrow$ *60*)

— Canonical lifting from predicates to state transformers and its simplification rules

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

lemma *p2ndf-simps*[*simp*]:
  $\lceil P \rceil \leq \lceil Q \rceil = (\forall\, s.\ P\ s \longrightarrow Q\ s)$
  $(\lceil P \rceil = \lceil Q \rceil) = (\forall\, s.\ P\ s = Q\ s)$
  $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s.\ P\ s \wedge Q\ s \rceil$
  $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s.\ P\ s \vee Q\ s \rceil$
  $tt\ \lceil P \rceil = \lceil P \rceil$
  $n\ \lceil P \rceil = \lceil \lambda s.\ \neg\ P\ s \rceil$
  unfolding *p2ndf-def one-nd-fun-def less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def*

  by (*auto simp*: *nd-fun-eq-iff kcomp-def le-fun-def n-op-nd-fun-def*)

— Meaning of the state-transformer Hoare triple

lemma *ndfun-kat-H*: $Hoare\ \lceil P \rceil\ X\ \lceil Q \rceil \longleftrightarrow (\forall\, s\ s'.\ P\ s \longrightarrow s' \in (X_\bullet)\ s \longrightarrow Q\ s')$
  unfolding *Hoare-def p2ndf-def less-eq-nd-fun-def times-nd-fun-def kcomp-def*
  by (*auto simp add*: *le-fun-def n-op-nd-fun-def*)

— Hoare triple for skip and a simp-rule

abbreviation *skip* $\equiv$ ($1{::}'a$ *nd-fun*)

lemma *H-skip*: $Hoare\ \lceil P \rceil\ skip\ \lceil P \rceil$
  using *H-skip* **by** *blast*

lemma *sH-skip*[*simp*]: $Hoare\ \lceil P \rceil\ skip\ \lceil Q \rceil \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
  unfolding *ndfun-kat-H* **by** (*simp add*: *one-nd-fun-def*)

— We introduce assignments and compute derive their rule of Hoare logic.

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

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

lemma *H-assign*: $P = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil$
  unfolding *ndfun-kat-H assign-def vec-upd-def* **by** *force*

lemma *sH-assign*[*simp*]: $Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil \longleftrightarrow (\forall\, s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j))$

unfolding *ndfun-kat-H vec-upd-def assign-def* **by** (*auto simp*: *fun-upd-def*)

— Next, the Hoare rule of the composition

abbreviation *seq-seq* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* (**infixl** ; *75*)
where $f \; ; \; g \equiv f \cdot g$

lemma *H-seq*: *Hoare* $\lceil P\rceil$ X $\lceil R\rceil \Longrightarrow$ *Hoare* $\lceil R\rceil$ Y $\lceil Q\rceil \Longrightarrow$ *Hoare* $\lceil P\rceil$ $(X \; ; \; Y)$ $\lceil Q\rceil$
by (*auto intro*: *H-seq*)

lemma *sH-seq*: *Hoare* $\lceil P\rceil$ $(X \; ; \; Y)$ $\lceil Q\rceil$ = *Hoare* $\lceil P\rceil$ (X) $\lceil\lambda s. \; \forall s'. \; s' \in (Y_\bullet) \; s \longrightarrow Q \; s'\rceil$
unfolding *ndfun-kat-H* **by** (*auto simp*: *times-nd-fun-def kcomp-def*)

— Rewriting the Hoare rule for the conditional statement

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

lemma *H-cond*: *Hoare* $\lceil\lambda s. \; P \; s \wedge B \; s\rceil$ X $\lceil Q\rceil \Longrightarrow$ *Hoare* $\lceil\lambda s. \; P \; s \wedge \neg \; B \; s\rceil$ Y $\lceil Q\rceil \Longrightarrow$
Hoare $\lceil P\rceil$ (*IF B THEN X ELSE Y*) $\lceil Q\rceil$
by (*rule H-cond*, *simp-all*)

lemma *sH-cond*[*simp*]: *Hoare* $\lceil P\rceil$ (*IF B THEN X ELSE Y*) $\lceil Q\rceil \longleftrightarrow$
(*Hoare* $\lceil\lambda s. \; P \; s \wedge B \; s\rceil$ X $\lceil Q\rceil \wedge$ *Hoare* $\lceil\lambda s. \; P \; s \wedge \neg \; B \; s\rceil$ Y $\lceil Q\rceil$)
by (*auto simp*: *H-cond-iff ndfun-kat-H*)

— Rewriting the Hoare rule for the while loop

abbreviation *while-inv-sugar* :: $'a$ *pred* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* (*WHILE - INV - DO -* [*64*,*64*,*64*] *63*)
where *WHILE B INV I DO X* $\equiv$ *kat-while-inv* $\lceil B\rceil$ $\lceil I\rceil$ X

lemma *sH-while-inv*: $\forall s. \; P \; s \longrightarrow I \; s \Longrightarrow \forall s. \; I \; s \wedge \neg \; B \; s \longrightarrow Q \; s \Longrightarrow$ *Hoare* $\lceil\lambda s. \; I \; s \wedge B \; s\rceil$ X $\lceil I\rceil$
$\Longrightarrow$ *Hoare* $\lceil P\rceil$ (*WHILE B INV I DO X*) $\lceil Q\rceil$
by (*rule H-while-inv*, *simp-all add*: *ndfun-kat-H*)

— Finally, we add a Hoare triple rule for finite iterations.

abbreviation *loopi-sugar* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *nd-fun* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP X INV I* $\equiv$ *kat-loop-inv* X $\lceil I\rceil$

lemma *H-loop*: *Hoare* $\lceil P\rceil$ X $\lceil P\rceil \Longrightarrow$ *Hoare* $\lceil P\rceil$ (*LOOP X INV I*) $\lceil P\rceil$
by (*auto intro*: *H-loop*)

lemma *H-loopI*: *Hoare* $\lceil I \rceil$ *X* $\lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow$ *Hoare* $\lceil P \rceil$ (*LOOP X INV I*) $\lceil Q \rceil$
using *H-loop-inv*[*of* $\lceil P \rceil$ $\lceil I \rceil$ *X* $\lceil Q \rceil$] **by** *auto*

0.13.2 Verification of hybrid programs

— Verification by providing evolution

definition *g-evol* :: $(('a{::}ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b\ pred \Rightarrow 'a\ set \Rightarrow 'b\ nd\text{-}fun$ (*EVOL*)
where $EVOL\ \varphi\ G\ T = (\lambda s.\ g\text{-}orbit\ (\lambda t.\ \varphi\ t\ s)\ G\ T)^{\bullet}$

lemma *H-g-evol*:
fixes $\varphi :: ('a{::}preorder) \Rightarrow 'b \Rightarrow 'b$
assumes $P = (\lambda s.\ (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows *Hoare* $\lceil P \rceil$ $(EVOL\ \varphi\ G\ T)$ $\lceil Q \rceil$
unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **using** *assms* **by** *clarsimp*

lemma *sH-g-evol*[*simp*]:
fixes $\varphi :: ('a{::}preorder) \Rightarrow 'b \Rightarrow 'b$
shows *Hoare* $\lceil P \rceil$ $(EVOL\ \varphi\ G\ T)$ $\lceil Q \rceil = (\forall s.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** *auto*

— Verification by providing solutions

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

lemma *H-g-orbital*:
$P = (\lambda s.\ (\forall X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \Longrightarrow$
Hoare $\lceil P \rceil$ $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q \rceil$
unfolding *ndfun-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

lemma *sH-g-orbital*: *Hoare* $\lceil P \rceil$ $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q \rceil =$
$(\forall s.\ P\ s \longrightarrow (\forall X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)))$
unfolding *g-orbital-eq g-ode-def ndfun-kat-H* **by** *auto*

context *local-flow*
begin

lemma *H-g-ode*:
assumes $P = (\lambda s.\ s \in S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows *Hoare* $\lceil P \rceil$ $(x' = f\ \&\ G\ on\ T\ S\ @\ 0)$ $\lceil Q \rceil$
proof(*unfold ndfun-kat-H g-ode-def g-orbital-eq assms*, *clarsimp*)

fix $s\ t\ X$
assume *hyps*: $t \in T$ $\forall x.\ x \in T \wedge x \leq t \longrightarrow G\ (X\ x)$ $X \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$
and *main*: $s \in S \longrightarrow (\forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
have $s \in S$
using *ivp-solsD*[*OF hyps*(*3*)] *init-time* **by** *auto*
hence $\forall \tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps* **by** *blast*
thus $Q\ (X\ t)$
using *main* ‹$s \in S$› *hyps* **by** *fastforce*
qed

lemma *sH-g-ode*: *Hoare* $\lceil P \rceil$ $(x{'}{=}\ f\ \&\ G\ on\ T\ S\ @\ 0)$ $\lceil Q \rceil$ =
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
fix $s\ t$
assume *hyps*: $s \in S$ $P\ s$ $t{\in}T$ $\forall \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)$
and *main*: $\forall s.\ P\ s \longrightarrow (\forall X{\in}Sols\ (\lambda t.\ f)\ T\ S\ 0\ s.\ \forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$
using *in-ivp-sols* **by** *blast*
thus $Q\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*
next
fix $s\ X\ t$
assume *hyps*: $P\ s$ $X \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$ $t \in T$ $\forall \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
and *main*: $\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
hence *obs*: $s \in S$
using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
hence $\forall \tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps* **by** *blast*
thus $Q\ (X\ t)$
using *hyps main obs* **by** *auto*
qed

lemma *sH-g-ode-ivl*: $\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow$ *Hoare* $\lceil P \rceil$ $(x{'}{=}\ f\ \&\ G\ on\ \{0..\tau\}\ S\ @\ 0)$ $\lceil Q \rceil$ =
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..\tau\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
fix $s\ t$
assume *hyps*: $0 \leq \tau$ $\tau \in T$ $s \in S$ $P\ s$ $t \in \{0..\tau\}$ $\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)$
and *main*: $\forall s.\ P\ s \longrightarrow (\forall X{\in}Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s.\ \forall t{\in}\{0..\tau\}.\ (\forall \tau'.\ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow G\ (X\ \tau')) \longrightarrow Q\ (X\ t))$
hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$
using *in-ivp-sols-ivl closed-segment-eq-real-ivl*[*of* $0\ \tau$] **by** *force*
thus $Q\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*

next
  fix *s X t*
  assume *hyps*: $0 \le \tau$ $\tau \in T$ $P\ s$ $X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$ $t \in \{0..\tau\}$
    $\forall \tau'.\ 0 \le \tau' \wedge \tau' \le \tau \wedge \tau' \le t \longrightarrow G\ (X\ \tau')$
    and *main*: $\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..\tau\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  hence $s \in S$
    using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
  have *obs1*: $\forall \tau{\in}down\ \{0..\tau\}\ t.\ D\ X = (\lambda t.\ f\ (X\ t))\ on\ \{0{-}{-}\tau\}$
    apply(*clarsimp*, *rule has-vderiv-on-subset*)
    using *ivp-solsD*(*1*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  have *obs2*: $X\ 0 = s$ $\forall \tau{\in}down\ \{0..\tau\}\ t.\ X \in \{0{-}{-}\tau\} \rightarrow S$
    using *ivp-solsD*(*2*,*3*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  have $\forall \tau{\in}down\ \{0..\tau\}\ t.\ \tau \in T$
 using *subintervalI*[*OF init-time* ‹$\tau \in T$›] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
  hence $\forall \tau{\in}down\ \{0..\tau\}\ t.\ X\ \tau = \varphi\ \tau\ s$
    using *obs1 obs2* **apply**(*clarsimp*)
    by (*rule eq-solution-ivl*) (*auto simp*: *closed-segment-eq-real-ivl*)
  thus $Q\ (X\ t)$
    using *hyps main* ‹$s \in S$› **by** *auto*
qed

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

end

— Verification with differential invariants

definition *g-ode-inv* :: $(('a{::}banach){\Rightarrow}'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow$
  $real \Rightarrow 'a\ pred \Rightarrow 'a\ nd\text{-}fun$ ((1x´=- & - on - - @ - DINV -))
  where ($x´= f$ & G *on* $T\ S$ @ t_0 *DINV* I) = ($x´= f$ & G *on* $T\ S$ @ t_0)

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

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

lemma *sH-diff-inv*[*simp*]: *Hoare* $\lceil I\rceil$ ($x´= f$ & G *on* $T\ S$ @ t_0) $\lceil I\rceil$ = *diff-invariant* $I\ f\ T\ S\ t_0\ G$
  unfolding *diff-invariant-eq ndfun-kat-H g-orbital-eq g-ode-def* **by** *auto*

lemma *H-g-ode-inv*: *Hoare* $\lceil I \rceil$ $(x' = f$ & G *on* $T\ S$ @ $t_0)$ $\lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow$

$\lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$ *Hoare* $\lceil P \rceil$ $(x' = f$ & G *on* $T\ S$ @ t_0 *DINV* $I)$ $\lceil Q \rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $q' = \lceil \lambda s.\ I\ s \wedge G\ s \rceil$ **in** *H-consr*, *simp*)
apply(*subst sH-g-orbital-guard*[*symmetric*], *force*)
by (*rule-tac* $I = I$ **in** *sH-g-orbital-inv*, *simp-all*)

0.13.3 Refinement Components

— Skip

lemma *R-skip*: $(\forall s.\ P\ s \longrightarrow Q\ s) \Longrightarrow 1 \leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
by (*auto simp*: *spec-def ndfun-kat-H one-nd-fun-def*)

— Composition

lemma *R-seq*: (*Ref* $\lceil P \rceil$ $\lceil R \rceil$) ; (*Ref* $\lceil R \rceil$ $\lceil Q \rceil$) $\leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
using *R-seq* **by** *blast*

lemma *R-seq-rule*: $X \leq$ *Ref* $\lceil P \rceil$ $\lceil R \rceil \Longrightarrow Y \leq$ *Ref* $\lceil R \rceil$ $\lceil Q \rceil \Longrightarrow X;\ Y \leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
unfolding *spec-def* **by** (*rule H-seq*)

lemmas *R-seq-mono* = *mult-isol-var*

— Assignment

lemma *R-assign*: $(x ::= e) \leq$ *Ref* $\lceil \lambda s.\ P\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j) \rceil$ $\lceil P \rceil$
unfolding *spec-def* **by** (*rule H-assign*, *clarsimp simp*: *fun-eq-iff fun-upd-def*)

lemma *R-assign-rule*: $(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
unfolding *sH-assign*[*symmetric*] *spec-def* **.**

lemma *R-assignl*: $P = (\lambda s.\ R\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow (x ::= e)$; *Ref* $\lceil R \rceil$ $\lceil Q \rceil \leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
apply(*rule-tac* $R = R$ **in** *R-seq-rule*)
by (*rule-tac R-assign-rule*, *simp-all*)

lemma *R-assignr*: $R = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow$ *Ref* $\lceil P \rceil$ $\lceil R \rceil$; $(x ::= e) \leq$ *Ref* $\lceil P \rceil$ $\lceil Q \rceil$
apply(*rule-tac* $R = R$ **in** *R-seq-rule*, *simp*)
by (*rule-tac R-assign-rule*, *simp*)

lemma $(x ::= e)$; *Ref* $\lceil Q \rceil$ $\lceil Q \rceil \leq$ *Ref* $\lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil$ $\lceil Q \rceil$
by (*rule R-assignl*) *simp*

lemma *Ref* $\lceil Q \rceil$ $\lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil$; $(x ::= e) \leq$ *Ref* $\lceil Q \rceil$ $\lceil Q \rceil$

by (*rule R-assignr*) *simp*

— Conditional

lemma *R-cond*: ($IF\ B\ THEN\ Ref\ \lceil\lambda s.\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil\ ELSE\ Ref\ \lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
using *R-cond*[*of* $\lceil B\rceil\ \lceil P\rceil\ \lceil Q\rceil$] **by** *simp*

lemma *R-cond-mono*: $X \leq X' \Longrightarrow Y \leq Y' \Longrightarrow (IF\ P\ THEN\ X\ ELSE\ Y) \leq IF\ P\ THEN\ X'\ ELSE\ Y'$
unfolding *kat-cond-def times-nd-fun-def plus-nd-fun-def n-op-nd-fun-def*
by (*auto simp*: *kcomp-def less-eq-nd-fun-def p2ndf-def le-fun-def*)

— While loop

lemma *R-while*: $WHILE\ Q\ INV\ I\ DO\ (Ref\ \lceil\lambda s.\ P\ s \wedge Q\ s\rceil\ \lceil P\rceil) \leq Ref\ \lceil P\rceil\ \lceil\lambda s.\ P\ s \wedge \neg\ Q\ s\rceil$
unfolding *kat-while-inv-def* **using** *R-while*[*of* $\lceil Q\rceil\ \lceil P\rceil$] **by** *simp*

lemma *R-while-mono*: $X \leq X' \Longrightarrow (WHILE\ P\ INV\ I\ DO\ X) \leq WHILE\ P\ INV\ I\ DO\ X'$
by (*simp add*: *kat-while-inv-def kat-while-def mult-isol mult-isor star-iso*)

— Finite loop

lemma *R-loop*: $X \leq Ref\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow LOOP\ X\ INV\ I \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \leq X' \Longrightarrow LOOP\ X\ INV\ I \leq LOOP\ X'\ INV\ I$
unfolding *kat-loop-inv-def* **by** (*simp add*: *star-iso*)

— Evolution command (flow)

lemma *R-g-evol*:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(EVOL\ \varphi\ G\ T) \leq Ref\ \lceil\lambda s.\ \forall t{\in}T.\ (\forall\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)\rceil\ \lceil P\rceil$
unfolding *spec-def* **by** (*rule H-g-evol*, *simp*)

lemma *R-g-evol-rule*:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall s.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (EVOL\ \varphi\ G\ T) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-evol*[*symmetric*] *spec-def* **.**

lemma *R-g-evoll*:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s.\ \forall t{\in}T.\ (\forall\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$

$(EVOL\ \varphi\ G\ T)$; $Ref\ \lceil R\rceil\ \lceil Q\rceil \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-g-evol-rule*, *simp-all*)

lemma *R-g-evolr*:
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$Ref\ \lceil P\rceil\ \lceil R\rceil;\ (EVOL\ \varphi\ G\ T) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-evol-rule*, *simp*)

lemma
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $EVOL\ \varphi\ G\ T$; $Ref\ \lceil Q\rceil\ \lceil Q\rceil \leq Ref\ \lceil \lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau$
$s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
by (*rule R-g-evoll*) *simp*

lemma
fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $Ref\ \lceil Q\rceil\ \lceil \lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil;\ EVOL$
$\varphi\ G\ T \leq Ref\ \lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-g-evolr*) *simp*

— Evolution command (ode)

context *local-flow*
begin

lemma *R-g-ode*: $(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq Ref\ \lceil \lambda s.\ s{\in}S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down$
$T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
unfolding *spec-def* **by** (*rule H-g-ode*, *simp*)

lemma *R-g-ode-rule*: $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q$
$(\varphi\ t\ s))) \Longrightarrow$
$(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode*[*symmetric*] **by** (*rule R2*)

lemma *R-g-odel*: $P = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x' = f\ \&\ G\ on\ T\ S\ @\ 0)$; $Ref\ \lceil R\rceil\ \lceil Q\rceil \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-g-ode-rule*, *simp-all*)

lemma *R-g-oder*: $R = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$

$Ref\ \lceil P\rceil\ \lceil R\rceil;\ (x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-ode-rule*, *simp*)

lemma $(x' = f\ \&\ G\ on\ T\ S\ @\ 0)$; $Ref\ \lceil Q\rceil\ \lceil Q\rceil \leq Ref\ \lceil \lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down$

T t. G $(\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
by (*rule R-g-odel*) *simp*

lemma *Ref* $\lceil Q\rceil\ \lceil\lambda s.\ \forall\, t{\in}T.\ (\forall\,\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$; $(x' = f$ & G *on* T S @ $0) \leq$ *Ref* $\lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-g-oder*) *simp*

lemma *R-g-ode-ivl*:
$\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow (\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}\{0..\tau\}.\ (\forall\,\tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = f$ & G *on* $\{0..\tau\}$ S @ $0) \leq$ *Ref* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode-ivl*[*symmetric*] **by** (*rule R2*)

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $I\ f\ T\ S\ t_0\ G \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil\lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$(x'{=}f$ & G *on* T S @ t_0 *DINV* $I) \leq$ *Ref* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *spec-def* **by** (*auto simp*: *H-g-ode-inv*)

0.13.4 Derivation of the rules of dL

We derive a generalised version of some domain specific rules of differential dynamic logic (dL).

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

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

lemma *diff-weak-rule*:
assumes $\lceil G\rceil \leq \lceil Q\rceil$
shows *Hoare* $\lceil P\rceil$ $(x' = f$ & G *on* T S @ $t_0)$ $\lceil Q\rceil$
using *assms* **unfolding** *g-orbital-eq ndfun-kat-H ivp-sols-def g-ode-def* **by** *auto*

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$

and *wp-C*:*Hoare* $\lceil P\rceil$ ($x\acute{}= f$ & G *on* T S @ t_0) $\lceil C\rceil$
and *wp-Q*:*Hoare* $\lceil P\rceil$ ($x\acute{}= f$ & (λ s. G s $\wedge$ C s) *on* T S @ t_0) $\lceil Q\rceil$
shows *Hoare* $\lceil P\rceil$ ($x\acute{}= f$ & G *on* T S @ t_0) $\lceil Q\rceil$
proof(*subst ndfun-kat-H*, *simp add*: *g-orbital-eq p2ndf-def g-ode-def*, *clarsimp*)
fix t::*real* **and** X::*real* $\Rightarrow$ *'a* **and** s **assume** P s **and** $t \in T$
and *x-ivp*:$X \in$ *ivp-sols* (λt. f) T S t_0 s
and *guard-x*:$\forall x$. $x \in T \wedge x \leq t \longrightarrow G$ (X x)
have $\forall$ $t$$\in$(*down* T t). X $t \in$ *g-orbital* f G T S t_0 s
using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
hence $\forall$ $t$$\in$(*down* T t). C (X t)
using *wp-C* ‹P s› **by** (*subst* (*asm*) *ndfun-kat-H*, *auto simp*: *g-ode-def*)
hence X $t \in$ *g-orbital* f (λs. G s $\wedge$ C s) T S t_0 s
using *guard-x* ‹$t \in T$› **by** (*auto intro*!: *g-orbitalI x-ivp*)
thus Q (X t)
using ‹P s› *wp-Q* **by** (*subst* (*asm*) *ndfun-kat-H*) (*auto simp*: *g-ode-def*)
qed

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

abbreviation *g-global-ode-inv* :: ((*'a*::*banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a nd-fun*
((*1x´=-* & - *DINV* -)) **where** ($x\acute{}= f$ & G *DINV* I) $\equiv$ ($x\acute{}= f$ & G *on UNIV UNIV* @ *0 DINV* I)

end

0.13.5 Examples

We prove partial correctness specifications of some hybrid systems with our refinement and verification components.

theory *kat2ndfun-examples*
imports *kat2ndfun*

begin

Pendulum

The ODEs x' t = y t and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use $s\$1$ to represent the x-coordinate and $s\$2$ for the y-coordinate. We prove that this motion remains circular.

abbreviation *fpend* :: *real^2* $\Rightarrow$ *real^2* (f)
where f s $\equiv$ (χ i. *if* i=*1* *then* $s\$2$ *else* $-s\$1$)

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

else $-$ *s*\$*1* $\cdot$ *sin* τ + *s*\$*2* $\cdot$ *cos* τ)

— Verified with annotated dynamics

lemma *pendulum-dyn*: *Hoare* $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (*EVOL* φ *G T*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by *simp*

— Verified with differential invariants

lemma *pendulum-inv*: *Hoare* $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (x'=*f* & *G*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified with the flow

lemma *local-flow-pend*: *local-flow f UNIV UNIV* φ
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *pendulum-flow*: *Hoare* $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ (x'=*f* & *G*) $\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*simp only*: *local-flow.sH-g-ode*[*OF local-flow-pend*], *simp*)

no-notation *fpend* (*f*)
and *pend-flow* (φ)

Bouncing Ball

A ball is dropped from rest at an initial height h. The motion is described with the free-fall equations $x'\ t = v\ t$ and $v'\ t = g$ where g is the constant acceleration due to gravity. The bounce is modelled with a variable assigntment that flips the velocity, thus it is a completely elastic collision with the ground. We use *s*\$*1* to ball's height and *s*\$*2* for its velocity. We prove that the ball remains above ground and below its initial resting position.

abbreviation *fball* :: *real* $\Rightarrow$ *real*ˆ*2* $\Rightarrow$ *real*ˆ*2* (*f*)
where *f g s* $\equiv$ (χ *i*. *if i=1 then s*\$*2 else g*)

abbreviation *ball-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*ˆ*2* $\Rightarrow$ *real*ˆ*2* (φ)
where φ *g* τ *s* $\equiv$ (χ *i*. *if i=1 then g* $\cdot$ τ ˆ *2*/*2* + *s*\$*2* $\cdot$ τ + *s*\$*1 else g* $\cdot$ τ + *s*\$*2*)

— Verified with differential invariants

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

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

lemma *fball-invariant*:
  fixes $g\ h$ :: *real*
  defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
  shows *diff-invariant I (f g) UNIV UNIV 0 G*
  unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
  by(*auto intro*!: *poly-derivatives*)

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *Hoare*
  $\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$
  (*LOOP*
      $((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2$
$= 0));$
       $(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ - s\$2))\ ELSE\ skip))$
    $INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
) $\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
  apply(*rule H-loopI*)
    apply(*rule H-seq*[**where** $R = \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot$
$s\$2$])
     apply(*rule H-g-ode-inv*)
  by (*auto simp*: *bb-real-arith intro*!: *poly-derivatives diff-invariant-rules*)

— Verified with annotated dynamics

lemma [*bb-real-arith*]:
  assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
    and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x{::}real) = 0$
  shows $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
    and $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
  from *pos* **have** $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
  then have $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$

```
    by (metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right
        monoid-mult-class.power2-eq-square semiring-class.distrib-left)
  hence g² · τ² + 2 · g · v · τ + v² + 2 · g · h = 0
    using invar by (simp add: monoid-mult-class.power2-eq-square)
  hence obs: (g · τ + v)² + 2 · g · h = 0
   apply(subst power2-sum) by (metis (no-types, hide-lams) Groups.add-ac(2, 3)

        Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2))
  thus 2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0
    by (simp add: monoid-mult-class.power2-eq-square)
  have  2 · g · h + (− ((g · τ) + v))² = 0
    using obs by (metis Groups.add-ac(2) power2-minus)
  thus 2 · g · h + (− (g · τ) − v) · (− (g · τ) − v) = 0
    by (simp add: monoid-mult-class.power2-eq-square)
qed

lemma [bb-real-arith]:
  assumes invar: 2 · g · x = 2 · g · h + v · v
  shows 2 · g · (g · τ² / 2 + v · τ + (x::real)) =
  2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (is ?lhs = ?rhs)
proof−
  have ?lhs = g² · τ² + 2 · g · v · τ + 2 · g · x
    by(auto simp: algebra-simps semiring-normalization-rules(29))
  also have ... = g² · τ² + 2 · g · v · τ + 2 · g · h + v · v (is ... = ?middle)
      by(subst invar, simp)
    finally have ?lhs = ?middle.
  moreover
  {have ?rhs = g · g · (τ · τ) + 2 · g · v · τ + 2 · g · h + v · 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-dyn: g < 0 ⟹ h ≥ 0 ⟹ Hoare
  ⌈λs. s$1 = h ∧ s$2 = 0⌉
  (LOOP
      ((EVOL (φ g) (λ s. s$1 ≥ 0) T);
       (IF (λ s. s$1 = 0) THEN (2 ::= (λs. − s$2)) ELSE skip))
    INV (λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
  ) ⌈λs. 0 ≤ s$1 ∧ s$1 ≤ h⌉
  apply(rule H-loopI, rule H-seq[where R=λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g ·
h + s$2 · s$2])
  by (auto simp: bb-real-arith)
```

— Verified with the flow

```
lemma local-flow-ball: local-flow (f g) UNIV UNIV (φ g)
```

apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def UNIV-2*)
by (*auto simp*: *forall-2 intro*!: *poly-derivatives*)

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *Hoare*
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$
(*LOOP*
$((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
$)\ \lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
apply(*rule H-loopI*)
apply(*rule H-seq*[**where** $R = \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$])
apply(*subst local-flow.sH-g-ode*[*OF local-flow-ball*])
apply(*force simp*: *bb-real-arith*)
by (*rule H-cond*) (*auto simp*: *bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g < (0::real) \Longrightarrow 0 \leq h \Longrightarrow$
$2 ::= (\lambda s. - s\$2) \leq Ref$
$\lceil \lambda s.\ s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
by (*rule R-assign-rule*, *auto*)

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows $Ref\ \lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil\ \lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil \geq$
(*LOOP*
$((EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
apply(*rule order-trans*)
apply(*rule R-loop-mono*) **defer**
apply(*rule R-loop*)
apply(*rule R-seq*)
using *assms* **apply**(*simp-all*, *force simp*: *bb-real-arith*)
apply(*rule R-seq-mono*) **defer**
apply(*rule order-trans*)
apply(*rule R-cond-mono*) **defer defer**
apply(*rule R-cond*) **defer**
using *R-bb-assign* **apply** *force*
apply(*rule R-skip*, *clarsimp*)
by (*rule R-g-evol-rule*, *force simp*: *bb-real-arith*)

no-notation *fball* (*f*)

and *ball-flow* (φ)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every τ minutes, it sets its chronometer to *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = - a * (T - U)$ where $U = L \geq 0$ when the heater is on, and $U = 0$ when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, and *3* is a variable to save temperature measurements. Finally, *4* states whether the heater is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *therm-vec-field* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (*f*)
  where *f a L s* ≡ (χ *i. if i* = *2 then 1 else* (*if i* = *1 then* − *a* ∗ (*s$1* − *L*) *else* *0*))

abbreviation *therm-guard* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*G*)
  where *G Tmin Tmax a L s* ≡ (*s$2* ≤ − (*ln* ((*L*−(*if L*=*0 then Tmin else Tmax*))/(*L*−*s$3*)))/*a*)

abbreviation *therm-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*I*)
  where *I Tmin Tmax s* ≡ *Tmin* ≤ *s$1* ∧ *s$1* ≤ *Tmax* ∧ (*s$4* = *0* ∨ *s$4* = *1*)

abbreviation *therm-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (φ)
  where φ *a L* τ *s* ≡ (χ *i. if i* = *1 then* − *exp*(−*a* ∗ τ) ∗ (*L* − *s$1*) + *L else*
  (*if i* = *2 then* τ + *s$2 else s$i*))

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \Longrightarrow \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$
proof(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)
  assume *a1*: *0* < *a*
  have *f2*: ⋀*r ra*. |(*r*::*real*) + − *ra*| = |*ra* + − *r*|
    by (*metis abs-minus-commute minus-real-def*)
  have ⋀*r ra rb*. (*r*::*real*) ∗ *ra* + − (*r* ∗ *rb*) = *r* ∗ (*ra* + − *rb*)
    by (*metis minus-real-def right-diff-distrib*)
  hence $|a * (s_1\$1 + - L) + - (a * (s_2\$1 + - L))| = a * |s_1\$1 + - s_2\$1|$
    using *a1* **by** (*simp add*: *abs-mult*)
  thus $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$
    using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-therm-dyn*:
  assumes *0* < (*a*::*real*)
  shows *local-lipschitz UNIV UNIV* (λ*t*::*real*. *f a L*)
  apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)

apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*)
using *assms* **apply**(*simp-all add*: *norm-diff-therm-dyn*)
apply(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *clarsimp*)
unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-therm*: $a > 0 \Longrightarrow$ *local-flow* (f a L) *UNIV UNIV* (φ a L)
by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-therm-dyn*
simp: *forall-4 vec-eq-iff*)

lemma *therm-dyn-down-real-arith*:
assumes $a > 0$ **and** *Thyps*: $0 < Tmin$ $Tmin \leq T$ $T \leq Tmax$
and *thyps*: $0 \leq (\tau$::*real*) $\forall\tau\in\{0..\tau\}$. $\tau \leq -$ (*ln* ($Tmin$ / T) / a)
shows $Tmin \leq exp\ (-a * \tau) * T$ **and** $exp\ (-a * \tau) * T \leq Tmax$
proof−
have $0 \leq \tau \wedge \tau \leq -$ (*ln* ($Tmin$ / T) / a)
using *thyps* **by** *auto*
hence *ln* ($Tmin$ / T) $\leq - a * \tau \wedge - a * \tau \leq 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have $Tmin$ / $T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $Tmin$ / $T \leq exp\ (-a * \tau)$ $exp\ (-a * \tau) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
thus $Tmin \leq exp\ (-a * \tau) * T$
using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
show $exp\ (-a * \tau) * T \leq Tmax$
using *Thyps mult-left-le-one-le*[*OF - exp-ge-zero obs*(*2*), *of T*]
less-eq-real-def order-trans-rules(*23*) **by** *blast*
qed

lemma *therm-dyn-up-real-arith*:
assumes $a > 0$ **and** *Thyps*: $Tmin \leq T$ $T \leq Tmax$ $Tmax < (L$::*real*)
and *thyps*: $0 \leq \tau$ $\forall\tau\in\{0..\tau\}$. $\tau \leq -$ (*ln* (($L - Tmax$) / ($L - T$)) / a)
shows $L - Tmax \leq exp\ (-(a * \tau)) * (L - T)$
and $L - exp\ (-(a * \tau)) * (L - T) \leq Tmax$
and $Tmin \leq L - exp\ (-(a * \tau)) * (L - T)$
proof−
have $0 \leq \tau \wedge \tau \leq -$ (*ln* (($L - Tmax$) / ($L - T$)) / a)
using *thyps* **by** *auto*
hence *ln* (($L - Tmax$) / ($L - T$)) $\leq - a * \tau \wedge - a * \tau \leq 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have ($L - Tmax$) / ($L - T$) > 0
using *Thyps* **by** *auto*
ultimately have ($L - Tmax$) / ($L - T$) $\leq exp\ (-a * \tau) \wedge exp\ (-a * \tau) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
moreover have $L - T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $(L - Tmax) \leq exp\ (-a * \tau) * (L - T) \wedge exp\ (-a * \tau)$
$* (L - T) \leq (L - T)$
by (*simp add*: *pos-divide-le-eq*)

thus $(L - Tmax) \le exp\ (-(a * \tau)) * (L - T)$
by *auto*
thus $L - exp\ (-(a * \tau)) * (L - T) \le Tmax$
by *auto*
show $Tmin \le L - exp\ (-(a * \tau)) * (L - T)$
using *Thyps* **and** *obs* **by** *auto*
qed

lemmas *H-g-ode-therm* = *local-flow.sH-g-ode-ivl*[*OF local-flow-therm - UNIV-I*]

lemma *thermostat-flow*:
assumes $0 < a$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Hoare* $\lceil I\ Tmin\ Tmax \rceil$
(*LOOP* (
— control
($2 ::= (\lambda s.\ 0)$);
($3 ::= (\lambda s.\ s\$1)$);
(*IF* ($\lambda s.\ s\$4 = 0 \wedge s\$3 \le Tmin + 1$) *THEN*
($4 ::= (\lambda s.1)$)
ELSE IF ($\lambda s.\ s\$4 = 1 \wedge s\$3 \ge Tmax - 1$) *THEN*
($4 ::= (\lambda s.0)$)
ELSE skip);
— dynamics
(*IF* ($\lambda s.\ s\$4 = 0$) *THEN*
($x' = f\ a\ 0\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0$)
ELSE
($x' = f\ a\ L\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ \{0..\tau\}\ UNIV\ @\ 0$))
) *INV I Tmin Tmax*)
$\lceil I\ Tmin\ Tmax \rceil$
apply(*rule H-loopI*)
apply(*rule-tac* $R = \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac* $R = \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac* $R = \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0$ **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp-all add*: *H-g-ode-therm*[*OF assms*(*1*,*2*)])+
using *therm-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *therm-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of - Tmax*] **by** *auto*

— Refined with the flow

lemma *R-therm-dyn-down*:
assumes $a > 0$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* $\lceil \lambda s.\ s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil \ge$
($x' = f\ a\ 0\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0$)
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *assms therm-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of - Tmax*] **by** *auto*

lemma *R-therm-dyn-up*:
assumes $a > 0$ **and** $0 \le \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows *Ref* $\lceil\lambda s.\ s\$4 \neq 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1\rceil$ $\lceil I\ Tmin\ Tmax\rceil \geq$
$(x\acute{} = f\ a\ L\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *assms therm-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*] **by** *auto*

lemma *R-therm-dyn*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* $\lceil\lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1\rceil$ $\lceil I\ Tmin\ Tmax\rceil \geq$
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN$
$(x\acute{} = f\ a\ 0\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
ELSE
$(x\acute{} = f\ a\ L\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ \{0..\tau\}\ UNIV\ @\ 0))$
apply(*rule order-trans*, *rule R-cond-mono*)
using *R-therm-dyn-down*[*OF assms*] *R-therm-dyn-up*[*OF assms*] **by** (*auto intro*!: *R-cond*)

lemma *R-therm-assign1*: *Ref* $\lceil I\ Tmin\ Tmax\rceil$ $\lceil\lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0\rceil$ $\geq (2 ::= (\lambda s.\ 0))$
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-assign2*:
Ref $\lceil\lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0\rceil$ $\lceil\lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1\rceil \geq (3 ::= (\lambda s.\ s\$1))$
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-ctrl*:
Ref $\lceil I\ Tmin\ Tmax\rceil$ $\lceil\lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1\rceil \geq$
$(2 ::= (\lambda s.\ 0))$;
$(3 ::= (\lambda s.\ s\$1))$;
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN$
$(4 ::= (\lambda s.1))$
$ELSE\ IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)\ THEN$
$(4 ::= (\lambda s.0))$
ELSE skip)
apply(*rule R-seq-rule*)+
apply(*rule R-therm-assign1*)
apply(*rule R-therm-assign2*)
apply(*rule order-trans*)
apply(*rule R-cond-mono*)
apply(*rule R-assign-rule*) **defer**
apply(*rule R-cond-mono*)
apply(*rule R-assign-rule*) **defer**
apply(*rule R-skip*) **defer**
apply(*rule order-trans*)
apply(*rule R-cond-mono*)
apply *force*
by (*rule R-cond*)+ *auto*

lemma *R-therm-loop*: *Ref* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil I\ Tmin\ Tmax \rceil$ $\geq$
(*LOOP*
Ref $\lceil I\ Tmin\ Tmax \rceil$ $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$;
Ref $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil$
INV I Tmin Tmax)
by (*intro R-loop R-seq*, *simp-all*)

lemma *R-thermostat-flow*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil I\ Tmin\ Tmax \rceil$ $\geq$
(*LOOP* (
— control
(2 ::= ($\lambda s.\ 0$));(3 ::= ($\lambda s.\ s\$1$));
(*IF* ($\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1$) *THEN*
(4 ::= ($\lambda s.1$))
ELSE IF ($\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1$) *THEN*
(4 ::= ($\lambda s.0$))
ELSE skip);
— dynamics
(*IF* ($\lambda s.\ s\$4 = 0$) *THEN*
($x' = f\ a\ 0$ & *G Tmin Tmax a 0 on* $\{0..\tau\}$ *UNIV* @ 0)
ELSE
($x' = f\ a\ L$ & *G Tmin Tmax a L on* $\{0..\tau\}$ *UNIV* @ 0))
) *INV I Tmin Tmax*)
by (*intro order-trans*[*OF - R-therm-loop*] *R-loop-mono*
R-seq-mono R-therm-ctrl R-therm-dyn[*OF assms*])

no-notation *therm-vec-field* (f)
and *therm-flow* (φ)
and *therm-guard* (G)
and *therm-loop-inv* (I)

Water tank

— Variation of Hespanha and [**?**]

abbreviation *tank-vec-field* :: *real* $\Rightarrow$ *real*^4 $\Rightarrow$ *real*^4 (f)
where $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

abbreviation *tank-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*^4 $\Rightarrow$ *real*^4 (φ)
where $\varphi\ k\ \tau\ s \equiv (\chi\ i.\ if\ i = 1\ then\ k * \tau + s\$1\ else$
$(if\ i = 2\ then\ \tau + s\$2\ else\ s\$i))$

abbreviation *tank-guard* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*^4 $\Rightarrow$ *bool* (G)
where $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*^4 $\Rightarrow$ *bool* (I)
where $I\ hmin\ hmax\ s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *tank-diff-inv* :: *real* ⇒ *real* ⇒ *real* ⇒ *real*ˆ*4* ⇒ *bool* (*dI*)
where *dI hmin hmax k s* ≡ $s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$
$hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

— Verified with the flow

lemma *local-flow-tank*: *local-flow* (*f k*) *UNIV UNIV* (φ *k*)
apply (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def*, *unfold UNIV-4*)
by (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff*)

lemma *tank-arith*:
assumes $0 \leq (\tau{::}real)$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau\in\{0..\tau\}.\ \tau \leq -((hmin - y) / c_o) \Longrightarrow hmin \leq y - c_o * \tau$
and $\forall \tau\in\{0..\tau\}.\ \tau \leq (hmax - y) / (c_i - c_o) \Longrightarrow (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) \cdot \tau + y$
and $y \leq hmax \Longrightarrow y - c_o \cdot \tau \leq hmax$
apply(*simp-all add*: *field-simps le-divide-eq assms*)
using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

lemmas *H-g-ode-tank* = *local-flow.sH-g-ode-ivl*[*OF local-flow-tank - UNIV-I*]

lemma *tank-flow*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *Hoare* ⌈*I hmin hmax*⌉
(*LOOP*
— control
(($2 ::= (\lambda s.0)$);($3 ::= (\lambda s.\ s\$1)$);
(*IF* ($\lambda s.\ s\$4 = 0 \wedge s\$3 \leq hmin + 1$) *THEN* ($4 ::= (\lambda s.1)$) *ELSE*
(*IF* ($\lambda s.\ s\$4 = 1 \wedge s\$3 \geq hmax - 1$) *THEN* ($4 ::= (\lambda s.0)$) *ELSE skip*));
— dynamics
(*IF* ($\lambda s.\ s\$4 = 0$) *THEN* ($x' = f\ (c_i - c_o)$ & *G hmax* $(c_i - c_o)$ *on* $\{0..\tau\}$ *UNIV* @ *0*)
ELSE ($x' = f\ (-c_o)$ & *G hmin* $(-c_o)$ *on* $\{0..\tau\}$ *UNIV* @ *0*)))
INV I hmin hmax) ⌈*I hmin hmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* $\wedge s\$2=0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* $\wedge s\$2=0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* $\wedge s\$2=0$ **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp-all add*: *H-g-ode-tank*[*OF assms*(*1*)])
using *assms tank-arith*[*OF - assms*(*2*,*3*)] **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:

0 $\leq$ τ $\Longrightarrow$ *diff-invariant* (*dI hmin hmax k*) (*f k*) {*0*..τ} *UNIV 0 Guard*
apply(*intro diff-invariant-conj-rule*)
apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
apply(*rule-tac* ν'=λt. *0* **and** μ'=λt. *1* **in** *diff-invariant-leq-rule*, *simp-all*)
apply(*rule-tac* ν'=λt. *0* **and** μ'=λt. *0* **in** *diff-invariant-leq-rule*, *simp-all*)
apply(*force intro*!: *poly-derivatives*)+
by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

lemma *tank-inv-arith1*:
assumes *0* $\leq$ (τ::*real*) **and** $c_o < c_i$ **and** *b*: *hmin* $\leq y_0$ **and** *g*: $\tau \leq$ (*hmax* $- y_0$) / ($c_i - c_o$)
shows *hmin* $\leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq$ *hmax*
proof−
have $(c_i - c_o) \cdot \tau \leq$ (*hmax* $- y_0$)
using *g assms*(*2*,*3*) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
thus $(c_i - c_o) \cdot \tau + y_0 \leq$ *hmax*
by *auto*
show *hmin* $\leq (c_i - c_o) \cdot \tau + y_0$
using *b assms*(*1*,*2*) **by** (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)
qed

lemma *tank-inv-arith2*:
assumes *0* $\leq$ (τ::*real*) **and** *0* $< c_o$ **and** *b*: $y_0 \leq$ *hmax* **and** *g*: $\tau \leq -$ ((*hmin* $- y_0$) / c_o)
shows *hmin* $\leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq$ *hmax*
proof−
have $\tau \cdot c_o \leq y_0 -$ *hmin*
using *g* ‹*0* $< c_o$› *pos-le-minus-divide-eq* **by** *fastforce*
thus *hmin* $\leq y_0 - c_o \cdot \tau$
by (*auto simp*: *mult.commute*)
show $y_0 - c_o \cdot \tau \leq$ *hmax*
using *b assms*(*1*,*2*) **by** (*smt mult-nonneg-nonneg*)

qed

lemma *tank-inv*:
assumes *0* $\leq \tau$ **and** *0* $< c_o$ **and** $c_o < c_i$
shows *Hoare* ⌈*I hmin hmax*⌉
(*LOOP*
— control
((*2* ::=(λs.*0*));(*3* ::=(λs. *s*\$*1*));
(*IF* (λs. *s*\$*4* = *0* ∧ *s*\$*3* $\leq$ *hmin* + *1*) *THEN* (*4* ::= (λs.*1*)) *ELSE*
(*IF* (λs. *s*\$*4* = *1* ∧ *s*\$*3* $\geq$ *hmax* − *1*) *THEN* (*4* ::= (λs.*0*)) *ELSE skip*));
— dynamics
(*IF* (λs. *s*\$*4* = *0*) *THEN*
(*x´*= *f* ($c_i - c_o$) & *G hmax* ($c_i - c_o$) *on* {*0*..τ} *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$)))
ELSE

(x´= f (−c_o) & G hmin (−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin hmax (−c_o)))))
INV I hmin hmax) ⌈*I hmin hmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R=λs. I hmin hmax s* ∧ *s$2=0* ∧ *s$3 = s$1* **in** *H-seq*)
apply(*rule-tac R=λs. I hmin hmax s* ∧ *s$2=0* ∧ *s$3 = s$1* **in** *H-seq*)
apply(*rule-tac R=λs. I hmin hmax s* ∧ *s$2=0* **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp*, *simp*)+
apply(*rule H-cond*, *rule H-g-ode-inv*)
using *assms tank-inv-arith1* **apply**(*force simp*: *tank-diff-inv*, *simp*, *clarsimp*)
apply(*rule H-g-ode-inv*)
using *assms tank-diff-inv*[*of* - −c_o *hmin hmax*] *tank-inv-arith2* **by** *auto*

— Refined with differential invariants

lemma *R-tank-inv*:
assumes *0* ≤ τ **and** *0* < c_o **and** c_o < c_i
shows *Ref* ⌈*I hmin hmax*⌉ ⌈*I hmin hmax*⌉ ≥
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0* ∧ *s$3* ≤ *hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s. s$4 = 0*) *THEN*
(x´= f (c_i−c_o) & G hmax (c_i−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin hmax (c_i−c_o)))
ELSE
(x´= f (−c_o) & G hmin (−c_o) on {0..τ} UNIV @ 0 DINV (dI hmin hmax (−c_o)))))
INV I hmin hmax) (**is** *LOOP* (*?ctrl*;*?dyn*) *INV* - ≤ *?ref*)
proof−
— First we refine the control.
let *?Icntrl* = λ*s. I hmin hmax s* ∧ *s$2 = 0* ∧ *s$3 = s$1*
and *?cond* = λ*s. s$4 = 0* ∧ *s$3* ≤ *hmin + 1*
have *ifbranch1*: *4* ::= (λ*s.1*) ≤ *Ref* ⌈λ*s. ?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF* (λ*s. s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*) ≤
Ref ⌈λ*s.* ¬ *?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch2*)
apply(*rule order-trans*, *rule R-cond-mono*) **defer defer**
by (*rule R-cond*) (*auto intro*!: *R-assign-rule R-skip*)
have *ifthenelse*: (*IF ?cond THEN ?branch1 ELSE ?branch2*) ≤ *Ref* ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉ (**is** *?ifthenelse* ≤ -)
by (*rule R-cond*)
have (*IF ?cond THEN* (*4* ::= (λ*s.1*)) *ELSE* (*IF* (λ*s. s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*)) ≤
Ref ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉

apply(*rule-tac y=?ifthenelse* **in** *order-trans*, *rule R-cond-mono*)
using *ifbranch1 ifbranch2 ifthenelse* **by** *auto*
hence *ctrl*: *?ctrl* $\leq$ *Ref* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$
apply(*rule-tac R=?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R=*λs*. I hmin hmax s* $\wedge$ *s\$2* $=$ *0* **in** *R-seq-rule*)
by (*auto intro*!: *R-assign-rule*)
— Then we refine the dynamics.
have *dynup*: (x'*=f* ($c_i - c_o$) & *G hmax* ($c_i - c_o$) *on* {*0..*τ} *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$))) $\leq$
Ref $\lceil\lambda s$*. s\$4* $=$ *0* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule R-g-ode-inv*[*OF tank-diff-inv*[*OF assms*(*1*)]])
using *assms* **by** (*auto simp*: *tank-inv-arith1*)
have *dyndown*: (x'*=f* ($-c_o$) & *G hmin* ($-c_o$) *on* {*0..*τ} *UNIV* @ *0 DINV* (*dI hmin hmax* ($-c_o$))) $\leq$
Ref $\lceil\lambda s$*. s\$4* $\neq$ *0* $\wedge$ *?Icntrl s*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule R-g-ode-inv*)
using *tank-diff-inv*[*OF assms*(*1*), *of* $-c_o$] *assms*
by (*auto simp*: *tank-inv-arith2*)
have *dyn*: *?dyn* $\leq$ *Ref* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
apply(*rule order-trans*, *rule R-cond-mono*)
using *dynup dyndown* **by** (*auto intro*!: *R-cond*)
— Finally we put everything together.
have *pre-pos*: $\lceil$*I hmin hmax*$\rceil$ $\leq$ $\lceil$*I hmin hmax*$\rceil$
by *simp*
have *inv-inv*: *Ref* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*Ref* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$) $\leq$ *Ref* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*I hmin hmax*$\rceil$
by (*rule R-seq*)
have *loopref*: *LOOP Ref* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*Ref* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$) *INV I hmin hmax* $\leq$ *?ref*
apply(*rule R-loop*)
using *pre-pos inv-inv* **by** *auto*
have *obs*: *?ctrl*;*?dyn* $\leq$ *Ref* $\lceil$*I hmin hmax*$\rceil$ $\lceil$*?Icntrl*$\rceil$; (*Ref* $\lceil$*?Icntrl*$\rceil$ $\lceil$*I hmin hmax*$\rceil$)
apply(*rule R-seq-mono*)
using *ctrl dyn* **by** *auto*
show *LOOP* (*?ctrl*;*?dyn*) *INV I hmin hmax* $\leq$ *?ref*
by (*rule order-trans*[*OF - loopref*], *rule R-loop-mono*[*OF obs*])
qed

no-notation *tank-vec-field* (f)
and *tank-flow* (φ)
and *tank-guard* (G)
and *tank-loop-inv* (I)
and *tank-diff-inv* (dI)

end

0.14 Verification components with Kleene Algebras

We create verification rules based on various Kleene Algebras.

theory *VC-diffKAD-KA*
imports
KAT-and-DRA.PHL-KAT
KAD.Modal-Kleene-Algebra
Transformer-Semantics.Kleisli-Quantale

begin

0.14.1 Hoare logic and refinement in KAT

Here we derive the rules of Hoare Logic and a refinement calculus in Kleene algebra with tests.

notation t ($\mathfrak{tt}$)

hide-const t

no-notation *ars-r* (r)
and *if-then-else* (*if - then - else - fi* [64,64,64] 63)
and *while* (*while - do - od* [64,64] 63)

context *kat*
begin

— Definitions of Hoare Triple

definition *Hoare* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (H) **where**
$H\ p\ x\ q \longleftrightarrow \mathfrak{tt}\ p \cdot x \leq x \cdot \mathfrak{tt}\ q$

lemma *H-consl*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons1* **by** *blast*

lemma *H-consr*: $\mathfrak{tt}\ q' \leq \mathfrak{tt}\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons2* **by** *blast*

lemma *H-cons*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ p' \Longrightarrow \mathfrak{tt}\ q' \leq \mathfrak{tt}\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$
by (*simp add*: *H-consl H-consr*)

— Skip program

lemma *H-skip*: $H\ p\ 1\ p$
by (*simp add*: *Hoare-def*)

— Sequential composition

lemma *H-seq*: $H\ p\ x\ r \Longrightarrow H\ r\ y\ q \Longrightarrow H\ p\ (x \cdot y)\ q$
by (*simp add*: *Hoare-def phl-seq*)

— Conditional statement

definition *kat-cond* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*if - then - else - fi* [*64*,*64*,*64*] *63*)
where
if p then x else y fi = ($\mathfrak{tt}\ p \cdot x + n\ p \cdot y$)

lemma *H-var*: $H\ p\ x\ q \longleftrightarrow \mathfrak{tt}\ p \cdot x \cdot n\ q = 0$
by (*metis Hoare-def n-kat-3 t-n-closed*)

lemma *H-cond-iff*: $H\ p$ (*if r then x else y fi*) $q \longleftrightarrow H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ q \wedge H\ (\mathfrak{tt}\ p \cdot n\ r)\ y\ q$
proof –
have $H\ p$ (*if r then x else y fi*) $q \longleftrightarrow \mathfrak{tt}\ p \cdot (\mathfrak{tt}\ r \cdot x + n\ r \cdot y) \cdot n\ q = 0$
by (*simp add*: *H-var kat-cond-def*)
also have ... $\longleftrightarrow \mathfrak{tt}\ p \cdot \mathfrak{tt}\ r \cdot x \cdot n\ q + \mathfrak{tt}\ p \cdot n\ r \cdot y \cdot n\ q = 0$
by (*simp add*: *distrib-left mult-assoc*)
also have ... $\longleftrightarrow \mathfrak{tt}\ p \cdot \mathfrak{tt}\ r \cdot x \cdot n\ q = 0 \wedge \mathfrak{tt}\ p \cdot n\ r \cdot y \cdot n\ q = 0$
by (*metis add-0-left no-trivial-inverse*)
finally show *?thesis*
by (*metis H-var test-mult*)
qed

lemma *H-cond*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ q \Longrightarrow H\ (\mathfrak{tt}\ p \cdot n\ r)\ y\ q \Longrightarrow H\ p$ (*if r then x else y fi*) q
by (*simp add*: *H-cond-iff*)

— While loop

definition *kat-while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - od* [*64*,*64*] *63*) **where**
while b do x od = $(\mathfrak{tt}\ b \cdot x)^\star \cdot n\ b$

definition *kat-while-inv* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*while - inv - do - od* [*64*,*64*,*64*] *63*) **where**
while p inv i do x od = *while p do x od*

lemma *H-exp1*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ q \Longrightarrow H\ p\ (\mathfrak{tt}\ r \cdot x)\ q$
using *Hoare-def n-de-morgan-var2 phl.ht-at-phl-export1* **by** *auto*

lemma *H-while*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ p \Longrightarrow H\ p$ (*while r do x od*) $(\mathfrak{tt}\ p \cdot n\ r)$
proof –
assume *a1*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ p$
have $\mathfrak{tt}\ (\mathfrak{tt}\ p \cdot n\ r) = n\ r \cdot \mathfrak{tt}\ p \cdot n\ r$
using *n-preserve test-mult* **by** *presburger*
then show *?thesis*
using *a1 Hoare-def H-exp1 conway.phl.it-simr phl-export2 kat-while-def* **by** *auto*
qed

lemma *H-while-inv*: $tt\ p \leq tt\ i \Longrightarrow tt\ i \cdot n\ r \leq tt\ q \Longrightarrow H\ (tt\ i \cdot tt\ r)\ x\ i \Longrightarrow H\ p\ (while\ r\ inv\ i\ do\ x\ od)\ q$
  by (*metis H-cons H-while test-mult kat-while-inv-def*)

— Finite iteration

lemma *H-star*: $H\ i\ x\ i \Longrightarrow H\ i\ (x^\star)\ i$
  unfolding *Hoare-def* **using** *star-sim2* **by** *blast*

lemma *H-star-inv*:
  assumes $tt\ p \leq tt\ i$ **and** $H\ i\ x\ i$ **and** $(tt\ i) \leq (tt\ q)$
  shows $H\ p\ (x^\star)\ q$
proof−
  have $H\ i\ (x^\star)\ i$
    using *assms*(*2*) *H-star* **by** *blast*
  hence $H\ p\ (x^\star)\ i$
    unfolding *Hoare-def* **using** *assms*(*1*) *phl-cons1* **by** *blast*
  thus *?thesis*
    unfolding *Hoare-def* **using** *assms*(*3*) *phl-cons2* **by** *blast*
qed

definition *kat-loop-inv* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*loop - inv -* [*64*,*64*] *63*)
  where $loop\ x\ inv\ i = x^\star$

lemma *H-loop*: $H\ p\ x\ p \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ p$
  unfolding *kat-loop-inv-def* **by** (*rule H-star*)

lemma *H-loop-inv*: $tt\ p \leq tt\ i \Longrightarrow H\ i\ x\ i \Longrightarrow tt\ i \leq tt\ q \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ q$
  unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

lemma *H-inv*: $tt\ p \leq tt\ i \Longrightarrow tt\ i \leq tt\ q \Longrightarrow H\ i\ x\ i \Longrightarrow H\ p\ x\ q$
  by (*rule-tac* $p'=i$ **and** $q'=i$ **in** *H-cons*)

lemma *H-inv-plus*: $tt\ i = i \Longrightarrow tt\ j = j \Longrightarrow H\ i\ x\ i \Longrightarrow\ H\ j\ x\ j \Longrightarrow\ H\ (i + j)\ x\ (i + j)$
  unfolding *Hoare-def* **using** *combine-common-factor*
  by (*smt add-commute add.left-commute distrib-left join.sup.absorb-iff1 t-add-closed*)

lemma *H-inv-mult*: $tt\ i = i \Longrightarrow tt\ j = j \Longrightarrow H\ i\ x\ i \Longrightarrow\ H\ j\ x\ j \Longrightarrow\ H\ (i \cdot j)\ x\ (i \cdot j)$
  unfolding *Hoare-def* **by** (*smt n-kat-2 n-mult-comm t-mult-closure mult-assoc*)

end

0.14.2 refinement KAT

class *rkat* = *kat* +
  fixes *Ref* :: $'a \Rightarrow 'a \Rightarrow 'a$
  assumes *spec-def*: $x \leq Ref\ p\ q \longleftrightarrow H\ p\ x\ q$

begin

lemma *R1*: $H\ p\ (Ref\ p\ q)\ q$
  using *spec-def* **by** *blast*

lemma *R2*: $H\ p\ x\ q \Longrightarrow x \leq Ref\ p\ q$
  by (*simp add*: *spec-def*)

lemma *R-cons*: $\mathtt{t}\ p \leq \mathtt{t}\ p' \Longrightarrow \mathtt{t}\ q' \leq \mathtt{t}\ q \Longrightarrow Ref\ p'\ q' \leq Ref\ p\ q$
proof –
  assume *h1*: $\mathtt{t}\ p \leq \mathtt{t}\ p'$ **and** *h2*: $\mathtt{t}\ q' \leq \mathtt{t}\ q$
  have $H\ p'\ (Ref\ p'\ q')\ q'$
    by (*simp add*: *R1*)
  hence $H\ p\ (Ref\ p'\ q')\ q$
    using *h1 h2 H-consl H-consr* **by** *blast*
  thus *?thesis*
    by (*rule R2*)
qed

— Abort and skip programs

lemma *R-skip*: $1 \leq Ref\ p\ p$
proof –
  have $H\ p\ 1\ p$
    by (*simp add*: *H-skip*)
  thus *?thesis*
    by (*rule R2*)
qed

lemma *R-zero-one*: $x \leq Ref\ 0\ 1$
proof –
  have $H\ 0\ x\ 1$
    by (*simp add*: *Hoare-def*)
  thus *?thesis*
    by (*rule R2*)
qed

lemma *R-one-zero*: $Ref\ 1\ 0 = 0$
proof –
  have $H\ 1\ (Ref\ 1\ 0)\ 0$
    by (*simp add*: *R1*)
  thus *?thesis*
    by (*simp add*: *Hoare-def join.le-bot*)
qed

— Sequential composition

lemma *R-seq*: (*Ref p r*) · (*Ref r q*) ≤ *Ref p q*
proof –
  have *H p* (*Ref p r*) *r* **and** *H r* (*Ref r q*) *q*
    by (*simp add*: *R1*)+
  hence *H p* ((*Ref p r*) · (*Ref r q*)) *q*
    by (*rule H-seq*)
  thus *?thesis*
    by (*rule R2*)
qed

— Conditional statement

lemma *R-cond*: *if v then* (*Ref* (tt *v* · tt *p*) *q*) *else* (*Ref* (*n v* · tt *p*) *q*) *fi* ≤ *Ref p q*
proof –
  have *H* (tt *v* · tt *p*) (*Ref* (tt *v* · tt *p*) *q*) *q* **and** *H* (*n v* · tt *p*) (*Ref* (*n v* · tt *p*) *q*) *q*
    by (*simp add*: *R1*)+
  hence *H p* (*if v then* (*Ref* (tt *v* · tt *p*) *q*) *else* (*Ref* (*n v* · tt *p*) *q*) *fi*) *q*
    by (*simp add*: *H-cond n-mult-comm*)
 thus *?thesis*
    by (*rule R2*)
qed

— While loop

lemma *R-while*: *while q do* (*Ref* (tt *p* · tt *q*) *p*) *od* ≤ *Ref p* (tt *p* · *n q*)
proof –
  have *H* (tt *p* · tt *q*) (*Ref* (tt *p* · tt *q*) *p*) *p*
    by (*simp-all add*: *R1*)
  hence *H p* (*while q do* (*Ref* (tt *p* · tt *q*) *p*) *od*) (tt *p* · *n q*)
    by (*simp add*: *H-while*)
  thus *?thesis*
    by (*rule R2*)
qed

— Finite iteration

lemma *R-star*: (*Ref i i*)* ≤ *Ref i i*
proof –
  have *H i* (*Ref i i*) *i*
    using *R1* **by** *blast*
  hence *H i* ((*Ref i i*)*) *i*
    using *H-star* **by** *blast*
  thus *Ref i i** ≤ *Ref i i*
    by (*rule R2*)
qed

lemma *R-loop*: $loop\ (Ref\ p\ p)\ inv\ i \leq Ref\ p\ p$
unfolding *kat-loop-inv-def* **by** (*rule R-star*)

— Invariants

lemma *R-inv*: $\mathsf{tt}\ p \leq \mathsf{tt}\ i \Longrightarrow \mathsf{tt}\ i \leq \mathsf{tt}\ q \Longrightarrow Ref\ i\ i \leq Ref\ p\ q$
using *R-cons* **by** *force*

end

no-notation *kat-cond* (*if - then - else - fi* [*64*,*64*,*64*] *63*)
and *kat-while* (*while - do - od* [*64*,*64*] *63*)
and *kat-while-inv* (*while - inv - do - od* [*64*,*64*,*64*] *63*)
and *kat-loop-inv* (*loop - inv -* [*64*,*64*] *63*)

0.14.3 Verification in AKA (KAD)

Here we derive verification components with weakest liberal preconditions based on antidomain Kleene algebra (or Kleene algebra with domain)

context *antidomain-kleene-algebra*
begin

— Sequential composition

declare *fbox-mult* [*simp*]

— Conditional statement

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

lemma *fbox-export1*: $ad\ p + |x]\ q = |d\ p \cdot x]\ q$
using *a-d-add-closure addual.ars-r-def fbox-def fbox-mult* **by** *auto*

lemma *fbox-cond* [*simp*]: $|if\ p\ then\ x\ else\ y\ fi]\ q = (ad\ p + |x]\ q) \cdot (d\ p + |y]\ q)$
using *aka-cond-def a-closure′ ads-d-def ans-d-def fbox-add2 fbox-export1* **by** *auto*

— Finite iteration

definition *aka-loop-inv* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*loop - inv -* [*64*,*64*] *63*)
where $loop\ x\ inv\ i = x^\star$

lemma *fbox-stari*: $d\ p \leq d\ i \Longrightarrow d\ i \leq |x]\ i \Longrightarrow d\ i \leq d\ q \Longrightarrow d\ p \leq |x^\star]\ q$
by (*meson dual-order.trans fbox-iso fbox-star-induct-var*)

lemma *fbox-loopi*: $d\ p \leq d\ i \Longrightarrow d\ i \leq |x]\ i \Longrightarrow d\ i \leq d\ q \Longrightarrow d\ p \leq |loop\ x\ inv\ i]\ q$
unfolding *aka-loop-inv-def* **using** *fbox-stari* **by** *blast*

— Invariants

lemma *fbox-frame*: $d\ p \cdot x \leq x \cdot d\ p \Longrightarrow d\ q \leq |x]\ r \Longrightarrow d\ p \cdot d\ q \leq |x]\ (d\ p \cdot d\ r)$
  using *dual.mult-isol-var fbox-add1 fbox-demodalisation3 fbox-simp* **by** *auto*

lemma *plus-inv*: $i \leq |x]\ i \Longrightarrow j \leq |x]\ j \Longrightarrow (i + j) \leq |x]\ (i + j)$
  by (*metis ads-d-def dka.dsr5 fbox-simp fbox-subdist join.sup-mono order-trans*)

lemma *mult-inv*: $d\ i \leq |x]\ d\ i \Longrightarrow d\ j \leq |x]\ d\ j \Longrightarrow (d\ i \cdot d\ j) \leq |x]\ (d\ i \cdot d\ j)$
  using *fbox-demodalisation3 fbox-frame fbox-simp* **by** *auto*

end

0.14.4 Relational model

We show that relations form Kleene Algebras (KAT and AKA).

interpretation *rel-uq*: *unital-quantale Id* $(O)\ \bigcap\ \bigcup\ (\cap)\ (\subseteq)\ (\subset)\ (\cup)\ \{\}$ *UNIV*
  by (*unfold-locales*, *auto*)

lemma *power-is-relpow*: *rel-uq.power X m* = *X* ^^ *m* **for** *X*::′*a rel*
proof (*induct m*)
  case *0* **show** *?case*
    by (*metis rel-uq.power-0 relpow.simps*(*1*))
  case *Suc* **thus** *?case*
    by (*metis rel-uq.power-Suc2 relpow.simps*(*2*))
qed

lemma *rel-star-def*: *X*^∗ = ($\bigcup$*m*. *rel-uq.power X m*)
  by (*simp add*: *power-is-relpow rtrancl-is-UN-relpow*)

lemma *rel-star-contl*: *X O Y*^∗ = ($\bigcup$*m*. *X O rel-uq.power Y m*)
by (*metis rel-star-def relcomp-UNION-distrib*)

lemma *rel-star-contr*: *X*^∗ *O Y* = ($\bigcup$*m*. (*rel-uq.power X m*) *O Y*)
  by (*metis rel-star-def relcomp-UNION-distrib2*)

interpretation *rel-ka*: *kleene-algebra* $(\cup)$ (*O*) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl*
proof
  fix *x y z* :: ′*a rel*
  show $Id \cup x\ O\ x^* \subseteq x^*$
    by (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)
next
  fix *x y z* :: ′*a rel*
  assume $z \cup x\ O\ y \subseteq y$
  thus $x^*\ O\ z \subseteq y$
    by (*simp only*: *rel-star-contr*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductl*)
next

fix $x\ y\ z :: 'a\ rel$
assume $z \cup y\ O\ x \subseteq y$
thus $z\ O\ x^* \subseteq y$
by (*simp only*: *rel-star-contl*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductr*)
qed

interpretation *rel-tests*: *test-semiring* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ $\lambda x.\ Id \cap (-x)$
by (*standard*, *auto*)

interpretation *rel-kat*: *kat* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl* $\lambda x.\ Id \cap (-x)$
by (*unfold-locales*)

definition *rel-R* $:: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ **where**
rel-R $P\ Q = \bigcup\{X.\ rel\text{-}kat.Hoare\ P\ X\ Q\}$

interpretation *rel-rkat*: *rkat* $(\cup)$ $(;)$ *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl* $(\lambda X.\ Id \cap -X)$ *rel-R*
by (*standard*, *auto simp*: *rel-R-def rel-kat.Hoare-def*)

lemma *RdL-is-rRKAT*: $(\forall x.\ \{(x,x)\};\ R1 \subseteq \{(x,x)\};\ R2) = (R1 \subseteq R2)$
by *auto*

definition *rel-ad* $:: 'a\ rel \Rightarrow 'a\ rel$ **where**
rel-ad $R = \{(x,x) \mid x.\ \neg (\exists y.\ (x,y) \in R)\}$

interpretation *rel-aka*: *antidomain-kleene-algebra rel-ad* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl*
by *unfold-locales* (*auto simp*: *rel-ad-def*)

0.14.5 State transformer model

We show that state transformers form Kleene Algebras (KAT and AKA).

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

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

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

lemma *nd-fun-eq-iff*: $(f = g) = (\forall x.\ (f_\bullet)\ x = (g_\bullet)\ x)$
by (*auto simp*: *nd-fun-ext*)

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

definition $0 = \zeta^\bullet$

definition *star-nd-fun f* = *qstar f* **for** f::'*a nd-fun*

definition $f + g = ((f_\bullet) \sqcup (g_\bullet))^\bullet$

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

lemma *nd-fun-plus-assoc*[*nd-fun-aka*]: $x + y + z = x + (y + z)$
and *nd-fun-plus-comm*[*nd-fun-aka*]: $x + y = y + x$
and *nd-fun-plus-idem*[*nd-fun-aka*]: $x + x = x$ **for** x::'*a nd-fun*
unfolding *plus-nd-fun-def* **by** (*simp add*: *ksup-assoc*, *simp-all add*: *ksup-comm*)

lemma *nd-fun-distr*[*nd-fun-aka*]: $(x + y) \cdot z = x \cdot z + y \cdot z$
and *nd-fun-distl*[*nd-fun-aka*]: $x \cdot (y + z) = x \cdot y + x \cdot z$ **for** x::'*a nd-fun*
unfolding *plus-nd-fun-def times-nd-fun-def* **by** (*simp-all add*: *kcomp-distr kcomp-distl*)

lemma *nd-fun-plus-zerol*[*nd-fun-aka*]: $0 + x = x$
and *nd-fun-mult-zerol*[*nd-fun-aka*]: $0 \cdot x = 0$
and *nd-fun-mult-zeror*[*nd-fun-aka*]: $x \cdot 0 = 0$ **for** x::'*a nd-fun*
unfolding *plus-nd-fun-def zero-nd-fun-def times-nd-fun-def* **by** *auto*

lemma *nd-fun-leq*[*nd-fun-aka*]: $(x \leq y) = (x + y = y)$
and *nd-fun-less*[*nd-fun-aka*]: $(x < y) = (x + y = y \wedge x \neq y)$
and *nd-fun-leq-add*[*nd-fun-aka*]: $z \cdot x \leq z \cdot (x + y)$ **for** x::'*a nd-fun*
unfolding *less-eq-nd-fun-def less-nd-fun-def plus-nd-fun-def times-nd-fun-def sup-fun-def*
by (*unfold nd-fun-eq-iff le-fun-def*, *auto simp*: *kcomp-def*)

lemma *nd-star-one*[*nd-fun-aka*]: $1 + x \cdot x^\star \leq x^\star$
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*]: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$ **for** x::'*a nd-fun*
unfolding *plus-nd-fun-def star-nd-fun-def*
apply(*simp-all add*: *fun-star-inductl sup-nd-fun.rep-eq fun-star-inductr*)
by (*metis order-refl sup-nd-fun.rep-eq uwqlka.conway.dagger-unfoldl-eq*)

instance
apply *intro-classes*
using *nd-fun-aka* **by** *simp-all*

end

instantiation *nd-fun* :: (*type*) *kat*
begin

definition $n\ f = (\lambda x.\ \mathit{if}\ ((f_\bullet)\ x = \{\})\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})^\bullet$

lemma *nd-fun-n-op-one*[*nd-fun-aka*]: $n\ (n\ (1$::'*a nd-fun*$)) = 1$
and *nd-fun-n-op-mult*[*nd-fun-aka*]: $n\ (n\ (n\ x \cdot n\ y)) = n\ x \cdot n\ y$
and *nd-fun-n-op-mult-comp*[*nd-fun-aka*]: $n\ x \cdot n\ (n\ x) = 0$

and *nd-fun-n-op-de-morgan*[*nd-fun-aka*]: $n\ (n\ (n\ x) \cdot n\ (n\ y)) = n\ x + n\ y$ **for** x::'*a nd-fun*
unfolding *n-op-nd-fun-def one-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*

by (*auto simp*: *nd-fun-eq-iff kcomp-def*)

instance
by (*intro-classes*, *auto simp*: *nd-fun-aka*)

end

instantiation *nd-fun* :: (*type*) *rkat*
begin

definition *Ref-nd-fun* $P\ Q \equiv (\lambda s.\ \bigcup\{(f_\bullet)\ s|f.\ Hoare\ P\ f\ Q\})^\bullet$

instance
apply(*intro-classes*)
by (*unfold Hoare-def n-op-nd-fun-def Ref-nd-fun-def times-nd-fun-def*)
(*auto simp*: *kcomp-def le-fun-def less-eq-nd-fun-def*)

end

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

definition $ad\ f = (\lambda x.\ if\ ((f_\bullet)\ x = \{\})\ then\ \{x\}\ else\ \{\})^\bullet$

lemma *nd-fun-ad-zero*[*nd-fun-aka*]: $ad\ x \cdot x = 0$
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*]: $ad\ (ad\ x) + ad\ x = 1$ **for** x::'*a nd-fun*
unfolding *antidomain-op-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*

by (*auto simp*: *nd-fun-eq-iff kcomp-def one-nd-fun-def*)

instance
apply *intro-classes*
using *nd-fun-aka* **by** *simp-all*

end

end

0.15 VC_diffKAD

theory *VC-diffKAD-auxiliarities*
imports *Main VC-diffKAD-KA Ordinary-Differential-Equations.ODE-Analysis*

begin

0.15.1 Stack Theories Preliminaries: VC_KAD and ODEs

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

type-synonym *'a pred* = *'a* ⇒ *bool*

type-synonym *'a store* = *string* ⇒ *'a*

hide-const η

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
  and *Archimedean-Field.floor* (⌊-⌋)
  and *Set.image* (‘)
  and *Range-Semiring.antirange-semiring-class.ars-r* (r)
  and *antidomain-semiringl.ads-d* (d)
  and *n-op* (n - [*90*] *91*)
  and *Hoare* (H)
  and *tau* (τ)
  and *dual* (∂)
  and *fres* (**infixl** ← *60*)
  and *n-add-op* (**infixl** ⊕ *65*)
  and *eta* (η)

notation *Set.image* (-⦇-⦈)
  and *Product-Type.prod.fst* (π_1)
  and *Product-Type.prod.snd* (π_2)
  and *List.zip* (**infixl** ⊗ *63*)
  and *rel-aka.fbox* (*wp*)

definition *p2r* :: *'a pred* ⇒ *'a rel* ((*1*⌈-⌉)) **where**
 $\lceil P\rceil = \{(s,s)\ |s.\ P\ s\}$

lemma *p2r-simps*[*simp*]:
 $\lceil P\rceil \leq \lceil Q\rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
 $(\lceil P\rceil = \lceil Q\rceil) = (\forall s.\ P\ s = Q\ s)$
 $(\lceil P\rceil\ ;\ \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \wedge Q\ s\rceil$
 $(\lceil P\rceil \cup \lceil Q\rceil) = \lceil \lambda\ s.\ P\ s \vee Q\ s\rceil$
 rel-ad $\lceil P\rceil = \lceil \lambda s.\ \neg\ P\ s\rceil$
 rel-aka.ads-d $\lceil P\rceil = \lceil P\rceil$
 unfolding *p2r-def rel-ad-def rel-aka.ads-d-def* **by** *auto*

lemma *wp-rel*: *wp* $R\ \lceil P\rceil = \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y\rceil$
 unfolding *rel-aka.fbox-def p2r-def rel-ad-def* **by** *auto*

lemma *boxProgrPred-chrctrztn*: $(x,y) \in$ *wp* $R\ \lceil P\rceil \longleftrightarrow (y = x \wedge (\forall y.\ (x,\ y) \in R \longrightarrow P\ y))$
 unfolding *wp-rel* **unfolding** *p2r-def* **by** *simp*

definition *assign* :: *string* ⇒ (*'a store* ⇒ *'a*) ⇒ (*'a store*) *rel* (- ::= -)
 where $(x ::= e) = \{(s,\ s(x := e\ s))\ |s.\ True\ \}$

lemma *wp-assign* [*simp*]: *wp* (*x* ::= *e*) $\lceil P\rceil = \lceil \lambda\ s.\ P\ (s(x := e\ s))\rceil$
unfolding *wp-rel assign-def* **by** *simp*

abbreviation *cond-sugar* :: *'a pred* ⇒ *'a rel* ⇒ *'a rel* ⇒ *'a rel* (*IF - THEN - ELSE -* [*64*,*64*] *63*)
where *IF P THEN X ELSE Y* ≡ *rel-aka.aka-cond* $\lceil P\rceil$ *X Y*

lemma *wp-loopI*: $\lceil P\rceil \le \lceil I\rceil \Longrightarrow \lceil I\rceil \le \lceil Q\rceil \Longrightarrow \lceil I\rceil \le wp\ R\ \lceil I\rceil \Longrightarrow \lceil P\rceil \le wp\ (R^*)\ \lceil Q\rceil$
using *rel-aka.fbox-stari*[*of* $\lceil P\rceil$ $\lceil I\rceil$] **by** *auto*

proposition *cons-eq-zipE*:
(x, y) # *tail* = *xList* ⊗ *yList* ⟹ ∃ *xTail yTail*. *x* # *xTail* = *xList* ∧ *y* # *yTail* = *yList*
by(*induction xList*, *simp-all*, *induction yList*, *simp-all*)

proposition *set-zip-left-rightD*:
$(x, y) \in$ *set* (*xList* ⊗ *yList*) ⟹ *x* ∈ *set xList* ∧ *y* ∈ *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*]

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

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

(primed) dSolve preliminaries

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

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

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

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

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

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

primrec *state-list-upd* :: ((*real* $\Rightarrow$ *real store* $\Rightarrow$ *real*) $\times$ *string* $\times$ (*real store* $\Rightarrow$ *real*)) *list* $\Rightarrow$
real $\Rightarrow$ *real store* $\Rightarrow$ *real store* **where**
state-list-upd [] *t s* = *s*|
state-list-upd (*uxf* # *tail*) *t s* = (*state-list-upd tail t s*)
((π_1 (π_2 *uxf*)) := (π_1 *uxf*) *t s*,
∂ (π_1 (π_2 *uxf*)) := (*if t* = *0 then* (π_2 (π_2 *uxf*)) *s*
else vderiv-of (λ *r*. (π_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*) (-[-←-] - [*64*,*64*,*64*] *63*)
where *s*[*xfList*←*uInput*] *t* $\equiv$ *state-list-upd* (*uInput* $\otimes$ *xfList*) *t s*

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

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

theorem *state-list-cross-upd-its-vars*:
assumes *distinctHyp*:$distinct\ (map\ \pi_1\ xfList)$
and *lengthHyp*:$length\ xfList = length\ uInput$
and *varsHyp*:$\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
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-on s g varDiffs*)(π_1 *xf* := *u 0 s*, ∂ (π_1 *xf*) := π_2 *xf s*) **by** *simp*
also have ... = (*override-on s g varDiffs*)(∂ (π_1 *xf*) := π_2 *xf s*)
using *override-on-def varDiffs-def assms* **by** *auto*
also have ... = (*override-on s* (*g*(∂ (π_1 *xf*) := π_2 *xf s*)) *varDiffs*)
using *observ* **and** *override-on-upd* **by** *force*
ultimately show *?thesis* **by** *auto*
qed

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

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

lemma *has-vderiv-on-cong-open-interval*:
assumes *gHyp*:$\forall$ τ > *0*. *f* τ = *g* τ **and** *tHyp*: *t*>*0*
and *fHyp*:(*f has-vderiv-on f'*) {*0*<..<*t*}
shows (*g has-vderiv-on f'*) {*0*<..<*t*}
proof−

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

lemma *inductive-to-sol-zero-its-dvars*:
assumes *eqFuncs*:$\forall\ s.\ \forall\, g.\ \forall\, xf{\in}set$ ((x, f) # *xfs*). π_2 *xf* (*override-on s g*
varDiffs) = π_2 *xf s*
and *eqLengths*:*length* ((x, f) # *xfs*) = *length* (u # *us*)
and *distinct*:*distinct* (*map* π_1 ((x, f) # *xfs*))
and *vars*:$\forall\, xf{\in}set$ ((x, f) # *xfs*). π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall\, uxf{\in}set$ ((u # *us*) $\otimes$ ((x, f) # *xfs*)). π_1 *uxf* 0 (*sol s*) = *sol s*
(π_1 (π_2 *uxf*))
and *disjHyp*:(y, g) = (x, f) $\vee$ (y, g) $\in$ *set xfs*
and *indHyp*:(y, g) $\in$ *set xfs* $\Longrightarrow$ (*sol s*[*xfs*$\leftarrow$*us*] 0) ($\partial\ y$) = g (*sol s*[*xfs*$\leftarrow$*us*] 0)
shows (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0) ($\partial\ y$) = g (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*]

0)
proof−
from *assms* **obtain** *h1* **where** *h1Def*:(*sol s*[((*x*, *f*) # *xfs*)←(*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*←*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[(*x*, *f*) # *xfs*←*u* # *us*] *0*)
proof
assume *eqHeads*:(*y*, *g*) = (*x*, *f*)
then have *g* (*sol s*[(*x*, *f*) # *xfs*←*u* # *us*] *0*) = *f* (*sol s*) **using** *h1Def eqFuncs* **by** *simp*
also have ... = (*sol s*[(*x*, *f*) # *xfs*←*u* # *us*] *0*) (∂ *y*) **using** *eqHeads* **by** *auto*
ultimately show *?thesis* **by** *linarith*
next
assume *tailHyp*:(*y*, *g*) ∈ *set xfs*
then have *y* ≠ *x* **using** *distinct set-zip-left-rightD* **by** *force*
hence ∂ *x* ≠ ∂ *y* **by**(*simp add*: *vdiff-def*)
have *x* ≠ ∂ *y* **using** *vars vdiff-invarDiffs* **by** *auto*
obtain *h2* **where** *h2Def*:(*sol s*[*xfs*←*us*] *0*) = *override-on* (*sol s*) *h2 varDiffs* **using** *state-list-cross-upd-its-dvars eqLengths distinct vars* **and** *solHyp1* **by** *force*
have (*sol s*[(*x*, *f*) # *xfs*←*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[*xfs*←*us*] *0*)
using *tailHyp indHyp* ‹*x* ≠ ∂ *y*› **and** ‹∂ *x* ≠ ∂ *y*› **by** *simp*
also have ... = *g* (*override-on* (*sol s*) *h2 varDiffs*) **using** *h2Def* **by** *simp*
also have ... = *g* (*sol s*) **using** *eqFuncs* **and** *tailHyp* **by** *force*
also have ... = *g* (*sol s*[(*x*, *f*) # *xfs*←*u* # *us*] *0*)
using *eqFuncs h1Def tailHyp* **and** *eq-snd-iff* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed

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

lemma *inductive-to-sol-greater-than-zero-its-dvars*:
assumes *lengthHyp*:*length* ((*y*, *g*) # *xfs*) = *length* (*v* # *vs*)
and *distHyp*:*distinct* (*map* π_1 ((*y*, *g*) # *xfs*))
and *varHyp*: ∀ *xf*∈*set* ((*y*, *g*) # *xfs*). π_1 *xf* ∉ *varDiffs*
and *indHyp*:(*u*,*x*,*f*) ∈ *set* (*vs* ⊗ *xfs*) ⟹ (*s*[*xfs*←*vs*]*t*)(∂ *x*) = *vderiv-of* (λ*r*. *u*

r s) {*0*<..<*2*$*_R$*t*} *t*
  and *disjHyp*:(*v*, *y*, *g*) = (*u*, *x*, *f*) ∨ (*u*, *x*, *f*) ∈ *set* (*vs* ⊗ *xfs*) **and** *tHyp*:*t* > *0*
 shows (*s*[(*y*, *g*) # *xfs*←*v* # *vs*] *t*) (∂ *x*) = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t*
proof−
 let *?lhs* = ((*s*[*xfs*←*vs*] *t*)(*y* := *v t s*, ∂ *y* := *vderiv-of* (λ *r*. *v r s*) {*0*<..< (*2* · *t*)} *t*)) (∂ *x*)
 let *?rhs* = *vderiv-of* (λ*r*. *u r s*) {*0*<..< (*2* · *t*)} *t*
 have (*s*[(*y*, *g*) # *xfs*←*v* # *vs*] *t*) (∂ *x*) = *?lhs* **using** *tHyp* **by** *simp*
 also have *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t* =*?rhs* **by** *simp*
 ultimately have *obs*:*?thesis* = (*?lhs* = *?rhs*) **by** *simp*
 from *disjHyp* **have** *?lhs* = *?rhs*
 proof
  assume *uxfEq*:(*v*, *y*, *g*) = (*u*, *x*, *f*)
  then have *?lhs* = *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* · *t*)} *t* **by** *simp*
  also have *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* · *t*)} *t* = *?rhs* **using** *uxfEq* **by** *simp*
  ultimately show *?lhs* = *?rhs* **by** *simp*
 next
  assume *sygTail*:(*u*, *x*, *f*) ∈ *set* (*vs* ⊗ *xfs*)
  from *this* **have** *y* ≠ *x* **using** *distHyp set-zip-left-rightD* **by** *force*
  hence ∂ *x* ≠ ∂ *y* **by**(*simp add*: *vdiff-def*)
  have *y* ≠ ∂ *x* **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
  then have *?lhs* = (*s*[*xfs*←*vs*] *t*) (∂ *x*) **using** ‹*y* ≠ ∂ *x*› **and** ‹∂ *x* ≠ ∂ *y*› **by** *simp*
  also have (*s*[*xfs*←*vs*] *t*) (∂ *x*) = *?rhs* **using** *indHyp sygTail* **by** *simp*
  ultimately show *?lhs* = *?rhs* **by** *simp*
 qed
 from *this* **and** *obs* **show** *?thesis* **by** *simp*
qed

lemma *to-sol-greater-than-zero-its-dvars*:
 assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
  and *lengthHyp*:*length xfList* = *length uInput*
  and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
  and *uxfHyp*:(*u*, *x*, *f*) ∈ *set* (*uInput* ⊗ *xfList*) **and** *tHyp*:*t* > *0*
 shows (*s*[*xfList*←*uInput*] *t*) (∂ *x*) = *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $*_R$ *t*)} *t*
 using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*, *clarify*)
 by(*rule-tac f*=*f* **in** *inductive-to-sol-greater-than-zero-its-dvars*, *auto*)

dInv preliminaries

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

no-notation *Antidomain-Semiring.antidomain-left-monoid-class.am-add-op* (**infixl** ⊕ *65*)
no-notation *Dioid.times-class.opp-mult* (**infixl** ⊙ *70*)
no-notation *Lattices.inf-class.inf* (**infixl** ⊓ *70*)
no-notation *Lattices.sup-class.sup* (**infixl** ⊔ *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*)

term *test-monoid-class.n-add-op*

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 ($\ominus$ ϑ) = *trmVars* ϑ|
trmVars ($\vartheta \oplus \eta$) = *trmVars* ϑ $\cup$ *trmVars* η|
trmVars ($\vartheta \odot \eta$) = *trmVars* ϑ $\cup$ *trmVars* η

fun *substList* ::(*string* $\times$ *trms*) *list* $\Rightarrow$ *trms* $\Rightarrow$ *trms* (-$\langle$-$\rangle$ [*54*] *80*) **where**
xtList$\langle t_C\ r\rangle = t_C\ r|$
$[]\langle t_V\ x\rangle = t_V\ x|$

$((y,\xi)$ # $xtTail)\langle Var\ x\rangle = (if\ x = y\ then\ \xi\ else\ xtTail\langle Var\ x\rangle)|$
$xtList\langle\ominus\ \vartheta\rangle = \ominus\ (xtList\langle\vartheta\rangle)|$
$xtList\langle\vartheta \oplus \eta\rangle = (xtList\langle\vartheta\rangle) \oplus (xtList\langle\eta\rangle)|$
$xtList\langle\vartheta \odot \eta\rangle = (xtList\langle\vartheta\rangle) \odot (xtList\langle\eta\rangle)$

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

ODE Extras

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

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

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

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

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

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

lemma *quadratic-monomial-derivative*:
$((\lambda t{::}real.\ a \cdot t^2)$ *has-derivative* $(\lambda t.\ a \cdot (2 \cdot x \cdot t)))$ $($*at x within T*$)$
apply$($*rule-tac* $g'1=\lambda\ t.\ 2 \cdot x \cdot t$ **in** *derivative-eq-intros*$(6))$
apply$($*rule-tac* $f'1=\lambda\ t.\ t$ **in** *derivative-eq-intros*$(16))$
by $($*auto intro*: *derivative-eq-intros*$)$

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

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

lemma *galilean-position*[*galilean-transform*]:
$((\lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x)$ *has-vderiv-on* $(\lambda t.\ a \cdot t + v))$ T
apply$($*rule-tac* $f'=\lambda\ x.\ a \cdot x + v$ **and** $g'1=\lambda\ x.\ 0$ **in** *derivative-intros*$(189))$
apply$($*rule-tac* $f'1=\lambda\ x.\ a \cdot x$ **and** $g'1=\lambda\ x.\ v$ **in** *derivative-intros*$(189))$
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*(*189*))
unfolding *has-vderiv-on-def* **by**(*auto intro*: *derivative-eq-intros*)

lemma [*galilean-transform-eq*]:
$t > 0 \Longrightarrow$ *vderiv-of* $(\lambda r.\ a \cdot r + v)\ \{0 <..< 2 \cdot t\}\ t = a$
proof–
let $?f =$ *vderiv-of* $(\lambda r.\ a \cdot r + v)\ \{0 <..< 2 \cdot t\}$
assume $t > 0$ **hence** $t \in \{0 <..< 2 \cdot t\}$ **by** *auto*
have $\exists\ f.\ ((\lambda r.\ a \cdot r + v)$ *has-vderiv-on* $f)\ \{0 <..< 2 \cdot t\}$
using *galilean-velocity* **by** *blast*
hence $((\lambda r.\ a \cdot r + v)$ *has-vderiv-on* $?f)\ \{0 <..< 2 \cdot t\}$

unfolding *vderiv-of-def* **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)
also have $((\lambda r.\ a \cdot r + v)$ *has-vderiv-on* $(\lambda t.\ a))\ \{0<..<2 \cdot t\}$
using *galilean-velocity* **by** *simp*
ultimately show (*vderiv-of* $(\lambda r.\ a \cdot r + v)\ \{0<..<2 \cdot t\})\ t = a$
apply(*rule-tac* $f'=?f$ **and** $\tau=t$ **and** $t=2\cdot t$ **in** *vderiv-unique-within-open-interval*)
using $\langle t \in \{0<..<2 \cdot t\}\rangle$ **by** *auto*
qed

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

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

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

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

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

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

begin

0.15.3 Phase Space Relational Semantics

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

$(y \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow \varphi_S\ t\ (\partial\ y) = 0)) \wedge$
— F solves the induced IVP.
$(\forall\ xf \in set\ xfList.\ ((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))\ solves\text{-}ode\ (\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))$
$\{0..t\}\ UNIV \wedge$
$\varphi_S\ 0\ (\pi_1\ xf) = s(\pi_1\ xf)))$

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

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

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

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

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

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$
and $y \notin (\pi_1(\!|set\ xfList|\!)) \cup varDiffs$
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* =
$\{(s,\varphi_S\ t)\ |s\ t\ \varphi_S.\ t \geq 0 \wedge (\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge solvesStoreIVP\ \varphi_S\ xfList$
$s\}$

abbreviation *vericond* :: $'a\ pred \Rightarrow\ 'a\ rel \Rightarrow\ 'a\ pred \Rightarrow bool$ (*PRE - - POST -*)
where $PRE\ P\ X\ POST\ Q \equiv \lceil P \rceil \subseteq wp\ X\ \lceil Q \rceil$

lemma *vericond-gevol*: (*PRE P* (*ODEsystem xfList with G*) *POST Q*) =
$(\forall\, s.\ P\ s \longrightarrow (\forall\, s'.\ (s,s') \in (ODEsystem\ xfList\ with\ G) \longrightarrow Q\ s'))$
unfolding *wp-rel* **by** *clarsimp*

0.15.4 Derivation of Differential Dynamic Logic Rules

"Differential Weakening"

lemma *wlp-evol-guard*:*Id* $\subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil G \rceil$
by(*simp add*: *rel-aka.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*
unfolding *wp-rel guarDiffEqtn-def* **using** *assms* **by** *auto*

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-aka.fbox-def rel-ad-def guarDiffEqtn-def*
by(*simp add*: *relcomp.simps p2r-def*, *fastforce*)

"Differential Cut"

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

lemma *condAfterEvol-remainsAlongEvol*:
assumes *boxDiffC*:$(s,\ s) \in$ *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*
proof(*clarsimp simp*: *wp-rel*)
fix *b y* **assume** *P b* **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 ‹*P b*› **by**(*clarsimp simp*: *wp-rel*)
then have $\forall\ r \in \{0..t\}.\ (b, \varphi_S\ r) \in$ (*ODEsystem xfList with* (λ *s*. *G s* $\wedge$ *C s*))
using $*$ *all-interval-guarDiffEqtn* **by** (*metis* (*mono-tags*, *lifting*))
from *this* **and** *pBoxCutQ* **have** $\forall\ r \in \{0..t\}.\ Q\ (\varphi_S\ r)$
using *boxProgrPred-chrctrztn* ‹*P b*› **by**(*clarsimp simp*: *wp-rel*)
thus *Q y*
using $*$ **by** *auto*
qed

theorem *dC*:
assumes *Id* $\subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil C \rceil$
shows *wp* (*ODEsystem xfList with G*) $\lceil Q \rceil$ = *wp* (*ODEsystem xfList with* (λ *s*. *G s* $\wedge$ *C s*)) $\lceil Q \rceil$
proof(*rule-tac f*=λ *x*. *wp x* $\lceil Q \rceil$ **in** *HOL.arg-cong*, *safe*)
fix *a b* **assume** (*a*, *b*) $\in$ *ODEsystem xfList with G*
then obtain φ_S *t* **where** $*$:*solvesStoreIVP* φ_S *xfList a* $\wedge$ ($\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge 0 \leq t \wedge \varphi_S\ t = b$
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* (*no-types*, *hide-lams*) *basic-trans-rules*(*31*) *pair-in-Id-conv*)
thus (*a*, *b*) $\in$ *ODEsystem xfList with* (λ*s*. *G s* $\wedge$ *C s*)
using $*$ *guarDiffEqtn-def* **by** *blast*
next
fix *a b* **assume** (*a*, *b*) $\in$ *ODEsystem xfList with* (λ*s*. *G s* $\wedge$ *C s*)
then show (*a*, *b*) $\in$ *ODEsystem xfList with G*
unfolding *guarDiffEqtn-def* **by**(*clarsimp*, *rule-tac x*=*t* **in** *exI*, *rule-tac x*=φ_S **in** *exI*, *simp*)
qed

Solve Differential Equation

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

from *this* **and** *uniqHyp* **have** $(sol\ s[xfList \leftarrow uInput]\ t) = \varphi_S\ t$ **by** *blast*
then have *cHyp*:$c = (sol\ s[xfList \leftarrow uInput]\ t)$ **using** *FHyp* **by** *simp*
from *this* **have** $G\ (sol\ s[xfList \leftarrow uInput]\ t)$ **using** *FHyp* **by** *force*
then show $Q\ c$ **using** *diffAssgn FHyp cHyp* **by** *auto*
qed

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

theorem *dSolve*:
assumes *solHyp*:$\forall\ s.\ solvesStoreIVP\ (\lambda t.\ sol\ s[xfList \leftarrow uInput]\ t)\ xfList\ s$
and *uniqHyp*:$\forall\ s.\ \forall\ X.\ solvesStoreIVP\ X\ xfList\ s \longrightarrow (\forall\ t \geq 0.(sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$
and *diffAssgn*: $\forall\ s.\ P\ s \longrightarrow (\forall\ t \geq 0.\ G\ (sol\ s[xfList \leftarrow uInput]\ t) \longrightarrow Q\ (sol\ s[xfList \leftarrow uInput]\ t))$
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*clarsimp*, *subgoal-tac a=b*)
apply(*clarify*, *subst boxProgrPred-chrctrztn*)
apply(*simp-all add*: *p2r-def*)
apply(*rule-tac uInput=uInput* **in** *prelim-dSolve*)
apply(*simp add*: *solHyp*, *simp add*: *uniqHyp*)
by (*metis* (*no-types*, *lifting*) *diffAssgn*)

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

lemma *conds4vdiffs-prelim*:
assumes *funcsHyp*:$\forall\ s\ g.\ \forall\ xf \in set\ xfList.\ \pi_2\ xf\ (override\text{-}on\ s\ g\ varDiffs) = \pi_2\ xf\ s$
and *distinctHyp*:$distinct\ (map\ \pi_1\ xfList)$
and *varsHyp*:$\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *lengthHyp*:$length\ xfList = length\ uInput$
and *solHyp1*:$\forall\ uxf \in set\ (uInput \otimes xfList).\ (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$
and *solHyp2*:$\forall\ t \geq 0.\ ((\lambda\tau.\ (sol\ s[xfList \leftarrow uInput]\ \tau)\ x)$

has-vderiv-on ($\lambda\tau$. *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*.
((λt. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λt. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0*..*t*}
and *uniqHyp*:$\forall$ *s*.$\forall$ *X*. *solvesStoreIVP X xfList s* $\longrightarrow$ ($\forall$ *t* $\geq$ *0*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = *X t*)
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolve*)
subgoal using *assms* **and** *conds4storeIVP-on-toSol* **by** *simp*
subgoal by (*simp add*: *uniqHyp*)
using *postCondHyp postCondHyp* **by** *simp*

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

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

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

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

then have *notInVars*:$x \notin (\pi_1 (\|set\ xfList\|)) \cup varDiffs$ **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-def*)

show $(sol\ s[xfList{\leftarrow}uInput]\ t)\ x = \varphi_s\ t\ x$
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-store-ivpD(3) xfHyp xDef* **by** *force*
also have $(sol\ s[xfList{\leftarrow}uInput]\ t)\ x = f\ (sol\ s[xfList{\leftarrow}uInput]\ t)$
using *solves-store-ivpD(3) xfHyp xDef solHyp tHyp* **by** *force*
ultimately show *?thesis* **using** *eqDerivs xfHyp tHyp* **by** *auto*
next case *False*
then have $\varphi_s\ t\ x = 0$
using *xDef solves-store-ivpD(2) Fsolves tHyp* **by** *simp*
also have $(sol\ s[xfList{\leftarrow}uInput]\ t)\ x = 0$
using *False solHyp tHyp solves-store-ivpD(2) xDef* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed

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

$(\lambda(t, (r{::}real)).\ (\pi_2\ xf)\ (sol\ s[xfList{\leftarrow}uInput]\ t))$
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-tac uInput=uInput* **in** *dSolve*)
prefer *2* **subgoal proof**(*clarify*)
fix *s*::*real store* **and** φ_s::*real* $\Rightarrow$ *real store* **and** *t*::*real*
assume *isSol*:*solvesStoreIVP* φ_s *xfList s* **and** *sHyp*:$0 \leq t$
from *this* **and** *uniqHyp* **have** $\forall\ xf \in set\ xfList.\ \forall\ t \in \{0..t\}.$
$(\pi_2\ xf)\ (\varphi_s\ t) = (\pi_2\ xf)\ (sol\ s[xfList{\leftarrow}uInput]\ t)$ **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-all*

theorem *dSolve-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*:$\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$

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

"Differential Invariant."

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

lemma *derivationLemma-baseCase*:
fixes *F*::*real* $\Rightarrow$ *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 $\forall\, t\geq 0.\ \forall\, r\in\{0..t\}.\ ((\lambda t.\ F\ t\ x)\ has\text{-}vector\text{-}derivative\ F\ r\ (\partial\ x))\ (at\ r\ within\ \{0..t\})$
proof(*cases* $x \in set\ (map\ \pi_1\ xfList)$)
case *True*
from *this* **and** *solves* **have** $\forall\ t \geq 0.\ \forall\, r\in\{0..t\}.\ \forall\ xf \in set\ xfList.$
$((\lambda\ t.\ F\ t\ (\pi_1\ xf))\ has\text{-}vector\text{-}derivative\ F\ r\ (\partial\ (\pi_1\ xf)))\ (at\ r\ within\ \{0..t\})$
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 $constD:\forall\ t \geq 0.\ \forall\, r\in\{0..t\}.\ ((\lambda\ r.\ a\ x)\ has\text{-}vector\text{-}derivative\ 0)\ (at\ r\ within\ \{0..t\})$
by (*auto intro*: *derivative-eq-intros*)
{fix $t\ r::real$
assume $t\geq 0$ **and** $r \in \{0..t\}$
hence $((\lambda\ s.\ a\ x)\ has\text{-}vector\text{-}derivative\ 0)\ (at\ r\ within\ \{0..t\})$ **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 $((\lambda\ s.\ F\ s\ x)\ has\text{-}vector\text{-}derivative\ 0)\ (at\ r\ within\ \{0..t\})$
using *has-vector-derivative-transform* **by** (*metis* $\langle r \in \{0..t\}\rangle$)**}**
hence $isZero:\forall\, t\geq 0.\forall\, r\in\{0..t\}.((\lambda\ t.\ F\ t\ x) has\text{-}vector\text{-}derivative\ 0)(at\ r\ within\ \{0..t\})$
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 $\forall\, r\in\{0..t\}.\ ((\lambda\ s.\ [\![\eta]\!]_t\ (F\ s)) has\text{-}vector\text{-}derivative\ [\![\partial_t\ \eta]\!]_t\ (F\ r))\ (at\ r\ within\ \{0..t\})$
using *termVarsHyp* **proof**(*induction* η)
case (*Const r*)
then show *?case* **by** *simp*
next
case (*Var y*)
then have $yHyp:y \in UNIV - varDiffs$ **by** *auto*
from *this tHyp* **and** *assms*(*1*) **show** *?case*

using *derivationLemma-baseCase* **by** *auto*
next
case (*Mns* η)
then show *?case*
apply(*clarsimp*)
by(*rule derivative-intros*, *simp*)
next
case (*Sum* $\eta 1$ $\eta 2$)
then show *?case*
apply(*clarsimp*)
by(*rule derivative-intros*, *simp-all*)
next
case (*Mult* $\eta 1$ $\eta 2$)
then show *?case*
apply(*clarsimp*)
apply(*subgoal-tac* $((\lambda s.\ [\![\eta 1]\!]_t\ (F\ s) *_R [\![\eta 2]\!]_t\ (F\ s))$ *has-vector-derivative* $[\![\partial_t\ \eta 1]\!]_t\ (F\ r) \cdot [\![\eta 2]\!]_t\ (F\ r) + [\![\eta 1]\!]_t\ (F\ r) \cdot [\![\partial_t\ \eta 2]\!]_t\ (F\ r))$ (*at* r *within* $\{0..t\}$),*simp*)
apply(*rule-tac* $f'1=[\![\partial_t\ \eta 1]\!]_t\ (F\ r)$ **and** $g'1=[\![\partial_t\ \eta 2]\!]_t\ (F\ r)$ **in** *derivative-eq-intros*(26))
by (*simp-all add*: *has-field-derivative-iff-has-vector-derivative*)
qed

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

and $3{:}\forall xf{\in}set\ ((y,\ g)\ \#\ xfTail).\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$
hence $*{:}[\![(map\ (vdiff \circ \pi_1)\ xfTail \otimes trmTail)\langle Var\ (\partial\ x)\rangle]\!]_t\ (F\ t) = F\ t\ (\partial\ x)$
using *tHyp* **by** *auto*
show $F\ t\ (\partial\ x) = [\![((map\ (vdiff \circ \pi_1)\ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V\ (\partial\ x)\rangle]\!]_t\ (F\ t)$
proof(*cases* $x \in set\ (map\ \pi_1\ ((y,\ g)\ \#\ xfTail))$)
case *True*
then have $x = y \vee (x \neq y \wedge x \in set\ (map\ \pi_1\ xfTail))$ **by** *auto*
moreover
{**assume** $x = y$

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

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

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

lemma *dInvForTrms-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$ $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta\rangle]\!]_t\ st = 0$
and *termVarsHyp*:*trmVars* $\eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a = 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a = 0$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
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*
hence *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 *tcHyp derivationLemma termVarsHyp* **by** *blast*
have $\forall r\in\{0..t\}.\ \forall\ xf \in set\ xfList.\ F\ r\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ r)$
using *tcHyp solves-store-ivpD*(*3*) **by** *fastforce*
hence $\forall r\in\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r\in\{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t\ \eta\rangle]\!]_t\ (F\ r) = 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** *fastforce*
ultimately have $\forall r\in\{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative* *0*) (*at r within* $\{0..t\}$)
using *obs2* **by** *auto*
hence $\forall s\in\{0..t\}.\ ((\lambda x.\ [\![\eta]\!]_t\ (F\ x))$ *has-derivative* $(\lambda x.\ x *_R 0))$ (*at s within* $\{0..t\}$)
using *tcHyp* **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 (*simp add*: *p2r-def*)
have $[\![\eta]\!]_t\ a = 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0)$
using *assms dInvForTrms-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0$
by *blast*
thus $(a,\ b) \in wp\ (ODEsystem\ xfList\ with\ G\ )\ \lceil\lambda s.\ [\![\eta]\!]_t\ s = 0\rceil$

using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)
qed

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

lemma *dInvForProps-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ st \geq 0$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a > 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c > 0)$
and $[\![\eta]\!]_t\ a \geq 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c \geq 0)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a > 0$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** $t::real$ **and** $F::real \Rightarrow real\ store$
where *tcHyp*:$t \geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a \wedge (\forall r \in \{0..t\}.\ G\ (F\ r))$
using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$
using *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$
using *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) > 0$
using *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))\ has\text{-}vector\text{-}derivative$
$[\![\partial_t\ \eta]\!]_t\ (F\ r))\ (at\ r\ within\ \{0..t\})$
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** $\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))\ has\text{-}derivative$
$(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))\ (at\ r\ within\ \{0..t\})$

by (*simp add*: *has-vector-derivative-def*)
hence $\exists\, r\in\{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![(\partial_t\ \eta)]\!]_t)\ (F\ r)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain r **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0 \wedge [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c > 0$
using *obs1 tcHyp* **by** (*smt mult-nonneg-nonneg*)
next
show $0 \leq [\![\eta]\!]_t\ a \longrightarrow (\forall\, c.\ (a,\ c) \in \mathit{ODEsystem}\ \mathit{xfList}\ \mathit{with}\ G \longrightarrow 0 \leq [\![\eta]\!]_t\ c)$
proof(*clarify*)
fix c **assume** *aHyp*:$[\![\eta]\!]_t\ a \geq 0$ **and** *cHyp*:$(a,\ c) \in \mathit{ODEsystem}\ \mathit{xfList}\ \mathit{with}\ G$
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t\geq 0 \wedge F\ t = c \wedge \mathit{solvesStoreIVP}\ F\ \mathit{xfList}\ a \wedge (\forall\, r\in\{0..t\}.\ G\ (F\ r))$
using *guarDiffEqtn-def* **by** *auto*
then have $\forall\, x.\ x \notin \mathit{varDiffs} \longrightarrow F\ 0\ x = a\ x$
using *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$
using *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) \geq 0$
using *aHyp tcHyp* **by** *simp*
hence *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 *tcHyp derivationLemma termVarsHyp* **by** *blast*
have $(\forall\, t\geq 0.\ \forall\ xf \in \mathit{set}\ \mathit{xfList}.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*
hence $\forall\, r\in\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall\, r\in\{0..t\}.\ [\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*$:$\forall\, r\in\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$
by *simp*
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* $\{0..t\}$)
using *obs2 tcHyp* **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** (*smt mult-nonneg-nonneg*)
qed
qed

lemma *less-pval-to-tval*:

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

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

lemma *dInv-prelim*:
assumes *substHyp*:$\forall\ st.\ G\ st \longrightarrow\ (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ \varphi\restriction]\!]_P\ st$
and *propVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\varphi]\!]_P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\varphi]\!]_P\ c)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\varphi]\!]_P\ a$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t\geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a$
using *guarDiffEqtn-def* **by** *auto*
from *aHyp propVarsHyp* **and** *substHyp* **show** $[\![\varphi]\!]_P\ c$
proof(*induction* φ)
case $(Eq\ \vartheta\ \eta)$
hence *hyp*:$\forall\, st.\ G\ st \longrightarrow\ (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ (\vartheta \doteq \eta)\restriction]\!]_P\ st$
by *blast*
then have $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t\ (\vartheta \oplus (\ominus\ \eta))\rangle]\!]_t\ st = 0$
by *simp*
also have $trmVars\ (\vartheta \oplus (\ominus\ \eta)) \subseteq UNIV - varDiffs$ **using** *Eq.prems*(*2*)
by *simp*
moreover have $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ a = 0$ **using** *Eq.prems*(*1*)
by *simp*
ultimately have $(\forall\, c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G\ \longrightarrow [\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ c = 0)$
using *dInvForTrms-prelim listsHyp* **by** *blast*
hence $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ (F\ t) = 0$
using *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t)$
by *simp*
also have $([\![\vartheta \doteq \eta]\!]_P)\ c = ([\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t))$
using *tcHyp* **by** *simp*
ultimately show *?case*
by *simp*
next
case $(Less\ \vartheta\ \eta)$

hence $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow 0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle \partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$
using *less-pval-to-tval* **by** *metis*
also from *Less.prems(2)* **have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$
by *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a > 0$
using *Less.prems(1)* **by** *simp*
ultimately have $(\forall\, c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c > 0)$
using *dInvForProps-prelim(1) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) > 0$
using *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\eta]\!]_t\ (F\ t) > [\![\vartheta]\!]_t\ (F\ t)$
by *simp*
also have $[\![\vartheta \prec \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) < [\![\eta]\!]_t\ (F\ t))$
using *tcHyp* **by** *simp*
ultimately show *?case*
by *simp*
next
case *(Leq* $\vartheta\ \eta$*)*
hence $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow 0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle \partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$
using *leq-pval-to-tval* **by** *metis*
also from *Leq.prems(2)* **have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$
by *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a \geq 0$
using *Leq.prems(1)* **by** *simp*
ultimately have $(\forall\, c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c \geq 0)$
using *dInvForProps-prelim(2) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) \geq 0$
using *tcHyp cHyp* **by** *simp*
from *this* **have** $([\![\eta]\!]_t\ (F\ t) \geq [\![\vartheta]\!]_t\ (F\ t))$
by *simp*
also have $[\![\vartheta \preceq \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) \leq [\![\eta]\!]_t\ (F\ t))$
using *tcHyp* **by** *simp*
ultimately show *?case*
by *simp*
next
case *(And* $\varphi 1\ \varphi 2$*)*
thus *?case*
by *simp*
next
case *(Or* $\varphi 1\ \varphi 2$*)*
thus *?case*
by *auto*
qed
qed

theorem *dInv*:
assumes $\forall$ *st*. *G st* $\longrightarrow$ ($\forall$ *str*. *str* $\notin$ (π_1(|*set xfList*|)) $\longrightarrow$ *st* (∂ *str*) = *0*) $\longrightarrow$ [[((*map* (*vdiff* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)$\upharpoonleft\partial_P$ $\varphi\upharpoonright$]]$_P$ *st*
and *termVarsHyp*:*propVars* $\varphi \subseteq$ (*UNIV* $-$ *varDiffs*)
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
and *phi-p*:*P* = [[φ]]$_P$
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 (*simp add*: *p2r-def*)
have *P a* $\longrightarrow$ ($\forall$ *c*. (*a*,*c*) $\in$ (*ODEsystem xfList with G*) $\longrightarrow$ *P c*)
using *assms dInv-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall$ *c*. (*a*,*c*) $\in$ (*ODEsystem xfList with G*) $\longrightarrow$ *P c*
by *blast*
thus (*a*, *b*) $\in$ *wp* (*ODEsystem xfList with G*) $\lceil P \rceil$
using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)
qed

theorem *dInvFinal*:
assumes $\forall$ *st*. *G st* $\longrightarrow$ ($\forall$ *str*. *str* $\notin$ (π_1(|*set xfList*|)) $\longrightarrow$ *st* (∂ *str*) = *0*) $\longrightarrow$ [[((*map* (*vdiff* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)$\upharpoonleft\partial_P$ $\varphi\upharpoonright$]]$_P$ *st*
and *termVarsHyp*:*propVars* $\varphi \subseteq$ (*UNIV* $-$ *varDiffs*)
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
and *impls*:$\lceil P \rceil \subseteq \lceil F \rceil \wedge \lceil F \rceil \subseteq \lceil Q \rceil$
and *phi-f*:*F* = [[φ]]$_P$
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac C*=[[φ]]$_P$ **in** *dCut*)
apply(*subgoal-tac* $\lceil F \rceil \subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil F \rceil$)
using *impls* **and** *phi-f* **apply** *blast*
apply(*subgoal-tac PRE F* (*ODEsystem xfList with G*) *POST F*, *simp*)
apply(*rule-tac* φ=φ **and** *uInput*=*uInput* **in** *dInv*)
prefer *5* **apply**(*subgoal-tac PRE P* (*ODEsystem xfList with* (λs. *G s* $\wedge$ *F s*)) *POST Q*, *simp add*: *phi-f*)
apply(*rule dWeakening*)
using *impls* **apply** *simp*
using *assms* **by** *simp-all*

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

begin

0.15.5 Rules Testing

In this section we test the recently developed rules with simple dynamical systems.

— Example of hybrid program verified with the rule dSolve and a single differential equation: $x' = v$.

lemma *motion-with-constant-velocity*:
  PRE $(\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' > 0)$
  $(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v''))]\ with\ (\lambda\ s.\ True))$
  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-rel 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*(*189*))
  apply (*rule-tac* $f'1=\lambda\ x.0$ **and** $g'1=\lambda\ x.1$ **in** *derivative-intros*(*192*))
  by (*auto intro*: *derivative-intros*)

Same hybrid program verified with dSolve and the system of ODEs: $x' = v, v' = a$. The uniqueness part of the proof requires a preliminary lemma.

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

lemma *motion-with-constant-acceleration*:
  PRE $(\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
  $(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]\ with\ (\lambda\ s.\ True))$
  POST $(\lambda\ s.\ (s\ ''y'' < s\ ''x''))$

apply(*rule-tac uInput*=[λ *t s. s ''a''* $\cdot$ *t* ^ *2/2* + *s ''v''* $\cdot$ *t* + *s ''x''*,
λ *t s. s ''a''* $\cdot$ *t* + *s ''v''*] **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal** **by**(*simp add*: *wp-rel vdiff-def add-strict-increasing2*)
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *clarify*, *rule conjI*)
by(*rule galilean-transform*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
by(*rule continuous-intros*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
subgoal for *s* φ_s *t r* **apply**(*rule flow-vel-is-galilean-vel*[*of* φ_s *''x''* - - - - *t*])
by(*simp-all add*: *varDiffs-def vdiff-def*)
apply(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*) **done**
by (*auto simp*: *varDiffs-def vdiff-def*)

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

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

lemma *single-hop-ball*:

PRE (λ *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* [($''x''$, λ *s*. *s* $''v''$),($''v''$,λ *s*. $-$ *s* $''g''$)] *with* (λ *s*. *0* $\leq$ *s* $''x''$)));
(*IF* (λ *s*. *s* $''x''$ = *0*) *THEN* ($''v''$ ::= (λ *s*. $-$ *c* $\cdot$ *s* $''v''$)) *ELSE* ($''v''$ ::= (λ *s*. *s* $''v''$))))
POST (λ *s*. *0* $\leq$ *s* $''x''$ $\wedge$ *s* $''x''$ $\leq$ *H*)
apply(*simp*, *subst dS*[*of* [λ *t s*. $-$ *s* $''g''$ $\cdot$ *t* ^ *2*/*2* + *s* $''v''$ $\cdot$ *t* + *s* $''x''$, λ *t s*. $-$ *s* $''g''$ $\cdot$ *t* + *s* $''v''$]])
— Given solution is actually a solution.
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton*, *safe*)
apply(*rule galilean-transform-eq*, *simp*)+
apply(*rule galilean-transform*)+
— Uniqueness of the flow.
apply(*rule ubcStoreUniqueSol*, *simp*)
apply(*simp add*: *vdiff-def del*: *comp-apply*)
apply(*auto intro*: *continuous-intros del*: *comp-apply*)[*1*]
apply(*rule continuous-intros*)+
apply(*simp add*: *vdiff-def*, *safe*)
apply(*clarsimp*) **subgoal for** *s X t* 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* $''x''$ $\geq$ *0*))
POST (λ *s*. *s* $''x''$ $\geq$ *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* $''x''$ $\geq$ *0*))
POST (λ *s*. *s* $''x''$ $\geq$ *0*)
apply(*simp add*: *wp-rel*)
by (*auto simp*: *relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def*)

— Example of system proved with a differential invariant.
lemma *circular-motion*:
PRE (λ *s*. (*s* $''x''$) $\cdot$ (*s* $''x''$) + (*s* $''y''$) $\cdot$ (*s* $''y''$) $-$ (*s* $''r''$) $\cdot$ (*s* $''r''$) = *0*)
(*ODEsystem* [($''x''$,(λ *s*. *s* $''y''$)),($''y''$,(λ *s*. $-$ *s* $''x''$))] *with G*)
POST (λ *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*'', ⊖ (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.

lemma *motion-with-constant-velocity-and-invariants*:
PRE (λ *s*. *s* ''*x*'' > *s* ''*y*'' ∧ *s* ''*v*'' > *0*)
(*ODEsystem* [(''*x*'', λ *s*. *s* ''*v*'')] *with* (λ *s*. *True*))
POST (λ *s*. *s* ''*x*''> *s* ''*y*'')
apply(*rule-tac C* = λ *s*. *s* ''*v*'' > *0* **in** *dCut*)
apply(*rule-tac* φ = (t_C *0*) ≺ (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* = λ *s*. *s* ''*x*'' > *s* ''*y*'' **in** *dCut*)
apply(*rule-tac* φ=(t_V ''*y*'') ≺ (t_V ''*x*'') **and** *uInput*=[t_V ''*v*''] **and**
F=λ *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 (λ *s*. *s* ''*y*'' < *s* ''*x*'' ∧ *s* ''*v*'' ≥ *0* ∧ *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 C* = λ *s*. *s* ''*a*'' > *0* **in** *dCut*)
apply(*rule-tac* φ = (t_C *0*) ≺ (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* = λ *s*. *s* ''*v*'' ≥ *0* **in** *dCut*)
apply(*rule-tac* φ = (t_C *0*) ⪯ (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* = λ *s*. *s* ''*x*'' > *s* ''*y*'' **in** *dCut*)
apply(*rule-tac* φ = (t_V ''*y*'') ≺ (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 (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*''))))

POST (λ *s.* $0 \leq s$ $''x'' \wedge s$ $''x'' \leq H$)
apply(*simp*, *rule-tac* $C = \lambda$ *s.* s $''g'' > 0$ **in** *dCut*)
apply(*rule-tac* φ = (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* φ = (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* φ = (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** $(v \cdot v)/(2 \cdot g) \geq 0$
using *divide-nonneg-pos* **by** *fastforce*
ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
PRE (λ *s.* $0 \leq s$ $''x'' \wedge s$ $''x'' = H \wedge s$ $''v'' = 0 \wedge s$ $''g'' > 0$)
((*ODEsystem* [($''x''$, λ *s.* s $''v''$),($''v''$,λ *s.* $- s$ $''g''$)] *with* (λ *s.* $0 \leq s$ $''x''$));
(*IF* (λ *s.* s $''x'' = 0$) *THEN* ($''v''$ ::= (λ *s.* $- s$ $''v''$)) *ELSE Id*))*
POST (λ *s.* $0 \leq s$ $''x'' \wedge s$ $''x'' \leq H$)
apply(*rule wp-loopI*[*of* - $\lambda s.$ $0 \leq s$ $''x'' \wedge 0 < s$ $''g'' \wedge$
$2 \cdot s$ $''g'' \cdot s$ $''x'' = 2 \cdot s$ $''g'' \cdot H - (s$ $''v'' \cdot s$ $''v'')$])
apply(*simp*, *clarsimp simp*: *bouncing-ball-invariant*, *simp*)
apply(*rule-tac* $C = \lambda$ *s.* s $''g'' > 0$ **in** *dCut*)
apply(*rule-tac* φ = ((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* φ = (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*)
by (*rule dWeakening*, *clarsimp*)

end