CPSVerification

CPSVerification

September 20, 2019

Contents

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

0.1.1 Functions

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

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

0.1.2 Orders

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

lemma *cSup-eq*:
  fixes $c{::}'a{::}conditionally\text{-}complete\text{-}lattice$
  assumes $\forall x \in X.\ x \leq c$ **and** $\exists x \in X.\ c \leq x$
  shows $Sup\ X = c$
  apply(*rule order-antisym*)
   apply(*rule cSup-least*)
  using *assms* **apply**(*blast*, *blast*)
  using *assms*(*2*) **apply** *safe*
  apply(*subgoal-tac* $x \leq Sup\ X$, *simp*)
  by (*metis assms*(*1*) *cSup-eq-maximum eq-iff*)

lemma *bdd-above-ltimes*:
  fixes $c{::}'a{::}linordered\text{-}ring\text{-}strict$
  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*

lemma *finite-nat-minimal-witness*:
  fixes $P :: ('a{::}finite) \Rightarrow nat \Rightarrow bool$
  assumes $\forall i.\ \exists N{::}nat.\ \forall n \geq N.\ P\ i\ n$

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

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*)

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 *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 *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 *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

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

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

m x s A]
assms f′ **by** (*auto simp*: *has-vector-derivative-def o-def*)
qed

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

declare *has-vderiv-on-const* [*poly-derivatives*]
and *has-vderiv-on-id* [*poly-derivatives*]
and *derivative-intros*(*191*) [*poly-derivatives*]
and *derivative-intros*(*192*) [*poly-derivatives*]
and *derivative-intros*(*194*) [*poly-derivatives*]

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

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

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

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

lemma *has-vderiv-on-mult-const*: $D\ (*)\ a = (\lambda x.\ a)\ on\ T$
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*(*15*)) *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-compose-eq*[*of* $\lambda t.\ cos\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
by(*auto intro*!: *derivative-eq-intros simp*: *fun-eq-iff*)

lemma *has-vderiv-on-sin-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-compose-eq*[*of* $\lambda t.\ sin\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
by(*auto intro*!: *derivative-eq-intros simp*: *fun-eq-iff*)

lemma *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-compose-eq*[*of* $\lambda t.\ exp\ t$])
by (*rule has-vderiv-on-exp*, *simp-all add*: *mult.commute*)

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

lemma *vderiv-npow-intro*[*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-compose-eq*[*of* $\lambda t.\ t\hat{}n$])
using *assms*(*1*) **apply**(*rule has-vderiv-on-power*)
using *assms* **by** *auto*

lemma *vderiv-cos-intro*[*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-sin-intro*[*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-exp-intro*[*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*

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

lemma *has-derivative-at-within-mono*:
  assumes $(t{::}real) \in interior\ T$ **and** $T \subseteq S$
    and $D\ f \mapsto f'\ at\ t\ within\ T$
  shows $D\ f \mapsto f'\ at\ t\ within\ S$
  using *assms*(*3*) **apply**(*unfold has-derivative-def tendsto-iff* , *safe*)

unfolding *netlimit-at-within-mono*[*OF assms*(*1*,*2*)] *netlimit-within-interior*[*OF assms*(*1*)]
by (*rule eventually-at-within-mono*[*OF assms*(*1*,*2*)]) *simp*

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

lemma *eventually-all-finite-mono*:
fixes P :: ($'a$::*finite*) $\Rightarrow$ $'b$ $\Rightarrow$ *bool*
assumes *h1*: $\forall\, i$. *eventually* (P i) F
and *h2*: $\forall\, x$. ($\forall\, i$. (P i x)) $\longrightarrow$ Q x
shows *eventually* Q F
proof−
have *eventually* (λx. $\forall\, i$. P i x) F
using *h1 eventually-all-finite2* **by** *blast*
thus *eventually* Q F
unfolding *eventually-def*
using *h2 eventually-mono* **by** *auto*
qed

0.1.6 Multivariable derivatives

lemma *frechet-vec-lambda*:
fixes f::*real* $\Rightarrow$ ($'a$::*banach*)^($'m$::*finite*) **and** x::*real* **and** T::*real set*
defines $x_0 \equiv$ *netlimit* (*at* x *within* T) **and** $m \equiv$ *real CARD*($'m$)
assumes $\forall\, i$. ((λy. ($f\ y$ \$ $i - f\,x_0$ \$ $i - (y - x_0) *_R f'\ x$ \$ i) $/_R$ ($\|y - x_0\|$)) $\longrightarrow$ *0*) (*at* x *within* T)
shows ((λy. ($f\ y - f\ x_0 - (y - x_0) *_R f'\ x$) $/_R$ ($\|y - x_0\|$)) $\longrightarrow$ *0*) (*at* x *within* T)
proof(*simp add*: *tendsto-iff*, *clarify*)
fix ε::*real* **assume** *0* $< \varepsilon$
let *?*Δ = λy. $y - x_0$ **and** *?*Δf = λy. $f\ y - f\ x_0$
let *?P* = $\lambda i\ e\ y$. *inverse* |*?*Δ y| $*$ ($\|f\,y$ \$ $i - f\,x_0$ \$ i − *?*Δ y $*_R$ $f'\ x$ \$ $i\|$) $< e$
and *?Q* = λy. *inverse* |*?*Δ y| $*$ ($\|$*?*Δf y − *?*Δ y $*_R$ $f'\ x\|$) $< \varepsilon$
have *0* $< \varepsilon$ / *sqrt m*
using ‹*0* $< \varepsilon$› **by** (*auto simp*: *assms*)
hence $\forall\, i$. *eventually* (λy. *?P* i (ε / *sqrt m*) y) (*at* x *within* T)

using *assms* **unfolding** *tendsto-iff* **by** *simp*
thus *eventually ?Q* (*at x within T*)
proof(*rule eventually-all-finite-mono*, *simp add*: *norm-vec-def L2-set-def*, *clarify*)
fix *t*::*real*
let $?c = inverse\ |t - x_0|$ **and** $?u\ t = \lambda i.\ f\ t\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ t *_R f'\ x\ \$\ i$
assume *hyp*:$\forall i.\ ?c * (\|?u\ t\ i\|) < \varepsilon\ /\ sqrt\ m$
hence $\forall i.\ (?c *_R (\|?u\ t\ i\|))^2 < (\varepsilon\ /\ sqrt\ m)^2$
by (*simp add*: *power-strict-mono*)
hence $\forall i.\ ?c^2 * ((\|?u\ t\ i\|))^2 < \varepsilon^2\ /\ m$
by (*simp add*: *power-mult-distrib power-divide assms*)
hence $\forall i.\ ?c^2 * ((\|?u\ t\ i\|))^2 < \varepsilon^2\ /\ m$
by (*auto simp*: *assms*)
also have $(\{\}::'m\ set) \neq UNIV \wedge finite\ (UNIV :: 'm\ set)$
by *simp*
ultimately have $(\sum i{\in}UNIV.\ ?c^2 * ((\|?u\ t\ i\|))^2) < (\sum (i::'m){\in}UNIV.\ \varepsilon^2\ /$
$m)$
by (*metis* (*lifting*) *sum-strict-mono*)
moreover have $?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) = (\sum i{\in}UNIV.\ ?c^2 * (\|?u\ t$
$i\|)^2)$
using *sum-distrib-left* **by** *blast*
ultimately have $?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) < \varepsilon^2$
by (*simp add*: *assms*)
hence $sqrt\ (?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2)) < sqrt\ (\varepsilon^2)$
using *real-sqrt-less-iff* **by** *blast*
also have ... $= \varepsilon$
using ‹$0 < \varepsilon$› **by** *auto*
moreover have $?c * sqrt\ (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) = sqrt\ (?c^2 * (\sum i{\in}UNIV.$
$(\|?u\ t\ i\|)^2))$
by (*simp add*: *real-sqrt-mult*)
ultimately show $?c * sqrt\ (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) < \varepsilon$
by *simp*
qed
qed

lemma *frechet-vec-nth*:
fixes $f::real \Rightarrow ('a::real\text{-}normed\text{-}vector)\hat{\ }'m$ **and** *x*::*real* **and** *T*::*real set*
defines $x_0 \equiv netlimit\ (at\ x\ within\ T)$
assumes $((\lambda y.\ (f\ y - f\ x_0 - (y - x_0) *_R f'\ x)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)\ (at\ x$
$within\ T)$
shows $((\lambda y.\ (f\ y\ \$\ i - f\ x_0\ \$\ i - (y - x_0) *_R f'\ x\ \$\ i)\ /_R\ (\|y - x_0\|)) \longrightarrow$
$0)\ (at\ x\ within\ T)$
proof(*unfold tendsto-iff dist-norm*, *clarify*)
let $?\Delta = \lambda y.\ y - x_0$ **and** $?\Delta f = \lambda y.\ f\ y - f\ x_0$
fix ε::*real* **assume** $0 < \varepsilon$
let $?P = \lambda y.\ \|(?\Delta f\ y - ?\Delta\ y *_R f'\ x)\ /_R\ (\|?\Delta\ y\|) - 0\| < \varepsilon$
and $?Q = \lambda y.\ \|(f\ y\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ y *_R f'\ x\ \$\ i)\ /_R\ (\|?\Delta\ y\|) - 0\| < \varepsilon$
have *eventually ?P* (*at x within T*)
using ‹$0 < \varepsilon$› *assms* **unfolding** *tendsto-iff* **by** *auto*
thus *eventually ?Q* (*at x within T*)

proof(*rule-tac P=?P* **in** *eventually-mono*, *simp-all*)
let *?u y i* = *f y* \$ *i* − *f* x_0 \$ *i* − *?*Δ *y* $*_R$ *f′ x* \$ *i*
fix *y* **assume** *hyp*:*inverse* |*?*Δ *y*| ∗ (‖*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*‖) < ε
have ‖(*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*) \$ *i*‖ ≤ ‖*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*‖
using *Finite-Cartesian-Product.norm-nth-le* **by** *blast*
also have ‖*?u y i*‖ = ‖(*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*) \$ *i*‖
by *simp*
ultimately have ‖*?u y i*‖ ≤ ‖*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*‖
by *linarith*
hence *inverse* |*?*Δ *y*| ∗ (‖*?u y i*‖) ≤ *inverse* |*?*Δ *y*| ∗ (‖*?*Δ*f y* − *?*Δ *y* $*_R$ *f′ x*‖)
by (*simp add*: *mult-left-mono*)
thus *inverse* |*?*Δ *y*| ∗ (‖*f y* \$ *i* − *f* x_0 \$ *i* − *?*Δ *y* $*_R$ *f′ x* \$ *i*‖) < ε
using *hyp* **by** *linarith*
qed
qed

lemma *has-derivative-vec-lambda*:
fixes *f*::*real* ⇒ (*′a*::*banach*)ˆ(*′n*::*finite*)
assumes ∀ *i*. *D* (λ*t*. *f t* \$ *i*) ↦ (λ *h*. *h* $*_R$ *f′ x* \$ *i*) (*at x within T*)
shows *D f* ↦ (λ*h*. *h* $*_R$ *f′ x*) *at x within T*
apply(*unfold has-derivative-def*, *safe*)
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
using *assms frechet-vec-lambda*[*of x T*] **unfolding** *has-derivative-def* **by** *auto*

lemma *has-derivative-vec-nth*:
assumes *D f* ↦ (λ*h*. *h* $*_R$ *f′ x*) *at x within T*
shows *D* (λ*t*. *f t* \$ *i*) ↦ (λ*h*. *h* $*_R$ *f′ x* \$ *i*) *at x within T*
apply(*unfold has-derivative-def*, *safe*)
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
using *frechet-vec-nth*[*of x T f*] *assms* **unfolding** *has-derivative-def* **by** *auto*

lemma *has-vderiv-on-vec-eq*[*simp*]:
fixes *x*::*real* ⇒ (*′a*::*banach*)ˆ(*′n*::*finite*)
shows (*D x* = *x′ on T*) = (∀ *i*. *D* (λ*t*. *x t* \$ *i*) = (λ*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* ⇒ *′a* ⇒ (*′a*::*real-normed-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 φ::*real* ⇒ *′a* ⇒ (*′a*::*real-normed-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 $Sols\ f\ T\ S\ t_0\ s = \{X\ |X.\ (D\ X = (\lambda t.\ f\ t\ (X\ t))\ on\ T) \wedge X\ t_0 = s \wedge X \in T \rightarrow S\}$

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 $\gamma\ X\ G\ T = \bigcup\{\mathcal{P}\ X\ (down\ T\ t)\ |t.\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}\}$

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

lemma $\gamma\ X\ (\lambda s.\ True)\ T = \{X\ t\ |t.\ t \in T\}$ **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 $g\text{-}orbital\ f\ G\ T\ S\ t_0\ s = \bigcup\{\gamma\ X\ G\ T\ |X.\ X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s\}$

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

lemma *g-orbital* $f\ G\ T\ S\ t_0\ s =$
$\{X\ t\ |t\ X.\ t \in T \wedge (D\ X = (f \circ X)\ on\ T) \wedge X\ t_0 = s \wedge X \in T \to S \wedge (\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\})\}$
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 μ::*'a*::*banach* $\Rightarrow$ *real*
assumes *Thyp*: *is-interval* $T\ t_0 \in T$
and $\forall X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (\forall \tau \in T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ T)$
shows *diff-invariant* $(\lambda s.\ \nu\ s \leq \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix $X\ \tau$ **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 $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \le \tau \Longrightarrow t < t_0$ $t \in T$
using ‹$t \in \{t_0 <\!--\!< \tau\}$› *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) -\ \nu'\ (X\ t)) \ge 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \le 0$
using *primed*(*1*,*2*)[*OF* ‹$t \in T$›] **by** *auto*
ultimately have $(\tau - t_0) * (\mu'\ (X\ t) -\ \nu'\ (X\ t)) \ge 0$
apply(*case-tac* $\tau \ge t_0$) **by** (*force*, *auto simp*: *split-mult-pos-le*)
hence $(\tau - t_0) * (\mu'\ (X\ t) -\ \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) \ge 0$
using *x-ivp*(*2*) **by** *auto*
hence $\nu\ (X\ \tau) \le \mu\ (X\ \tau)$
using *mvt* **by** *simp*}
thus $\nu\ (X\ \tau) \le \mu\ (X\ \tau)$
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 $\forall\ X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (\forall \tau{\in}T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \ge \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \le \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ T)$
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*: $\bigwedge\tau.\ \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \ge \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \le \nu'\ (X\ \tau)$
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 $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \le \tau \Longrightarrow t < t_0$ $t \in T$
using ‹$t \in \{t_0 <\!--\!< \tau\}$› *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) -\ \nu'\ (X\ t)) \ge 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) -$

ν' $(X\ t)) \leq 0$
  using *primed(1,2)[OF ‹t ∈ 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-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* ⇒ ('*a*::{*heine-borel*,*banach*}) ⇒ '*a* **and** T::*real set* **and** S::'*a set*
and t_0::*real*
 assumes *open-domain*: *open T open S*
  and *interval-time*: *is-interval T*
  and *init-time*: $t_0 \in T$
  and *cont-vec-field*: $\forall s \in S.$ *continuous-on* $T\ (\lambda t.\ f\ t\ s)$
  and *lipschitz-vec-field*: *local-lipschitz T S f*
begin

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

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

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

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

end

lemma *local-lipschitz-add*:
fixes *f1 f2* :: *real* $\Rightarrow$ *'a::banach* $\Rightarrow$ *'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*:
$\bigwedge\ t\ s.\ t \in T \Longrightarrow s \in S \Longrightarrow D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}t\}$
$\bigwedge\ s.\ s \in S \Longrightarrow \varphi\ 0\ s = s$

$\bigwedge t\ s.\ t \in T \Longrightarrow s \in S \Longrightarrow (\lambda t.\ \varphi\ t\ s) \in \{0{-}{-}t\} \rightarrow S$
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 $ex\text{-}ivl\ s = T$
using *existence-ivl-subset*[*of s*] **apply** *safe*
unfolding *existence-ivl-def csols-eq*
using *in-ivp-sols-ivl*[*OF - assms*] **by** *blast*

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

unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **using** *rHyp* ‹$t > 0$› **by** *simp*
moreover have $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ (at\ t\ within\ \{0{<}{-}{-}{<}t + r/2\})$
using *vderiv calculation* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *blast*
moreover have *open* $\{0{<}{-}{-}{<}t + r/2\}$
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **by** *simp*
ultimately show *?thesis*
using *subs that* **by** *blast*
qed

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

lemma *has-derivative-on-open3*:
assumes $s \in S$
obtains B **where** $0 \in B$ **and** *open* B **and** $B \subseteq T$

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

lemma *has-derivative-on-open*:
assumes $t \in T$ $s \in S$
obtains B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ *at t within B*
apply(*subgoal-tac* $t < 0 \lor t = 0 \lor t > 0$)
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 (λτ. φ (τ + t) s) ∈ Sols (λt. f) T S 0 (φ (0 + t) s)
  apply(rule ivp-solsI, rule vderiv-on-compose-add)
  using has-vderiv-on-domain has-vderiv-on-subset assms apply blast
  using in-domain assms by auto

lemma is-monoid-action:
  assumes s ∈ S and T = UNIV
  shows φ 0 s = s and φ (t₁ + t₂) s = φ t₁ (φ t₂ s)
proof−
  show φ 0 s = s
    using ivp assms by simp
  have φ (0 + t₂) s = φ t₂ s
    by simp
  also have φ t₂ s ∈ S
    using in-domain assms by auto
  finally show φ (t₁ + t₂) s = φ t₁ (φ t₂ s)
    using eq-solution[OF additive-in-ivp-sols] assms by auto
qed

definition orbit :: 'a ⇒ 'a set (γ^φ)
  where γ^φ s = g-orbital f (λs. True) T S 0 s

lemma orbit-eq[simp]:
  assumes s ∈ S
  shows γ^φ s = {φ t s| t. t ∈ 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 ∈ S
  shows g-orbital f G T S 0 s = {φ t s| t. t ∈ T ∧ (∀τ∈down T t. G (φ τ s))}
proof(rule subset-antisym, simp-all only: subset-eq)
  let ?gorbit = {φ t s |t. t ∈ T ∧ (∀τ∈down T t. G (φ τ s))}
  {fix s' assume s' ∈ g-orbital f G T S 0 s
    then obtain X and t where x-ivp:X ∈ Sols (λt. f) T S 0 s
      and X t = s' and t ∈ T and guard:(𝒫 X (down T t) ⊆ {s. G s})
      unfolding g-orbital-def g-orbit-eq by auto
    have obs:∀τ∈(down T t). X τ = φ τ s
      using eq-solution[OF x-ivp - assms] by blast
    hence 𝒫 (λt. φ t s) (down T t) ⊆ {s. G s}
      using guard by auto
    also have φ t s = X t
      using eq-solution[OF x-ivp ‹t ∈ T› assms] by simp
    ultimately have s' ∈ ?gorbit
      using ‹X t = s'› ‹t ∈ T› by auto}
  thus ∀s'∈ g-orbital f G T S 0 s. s' ∈ ?gorbit
    by blast
next
  let ?gorbit = {φ t s |t. t ∈ T ∧ (∀τ∈down T t. G (φ τ s))}
```

{fix *s'* **assume** *s'* $\in$ *?gorbit*
then obtain *t* **where** $\mathcal{P}$ ($\lambda t.$ φ *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* (λ *s*. *c*) *T UNIV* (λ *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=*λ *s*. *0* **and** *g'1=*λ *s*. *c* **in** *derivative-intros*(*191*))
apply(*rule derivative-intros*, *simp*)+
by *simp-all*

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

begin

Chapter 1

Linear Algebra for Hybrid Systems

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

1.1 Vector operations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

unfolding *column-def vec-eq-iff* **by** *simp*
qed

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

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

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

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

1.2 Matrix norms

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

1.2.1 Matrix operator norm

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

lemma *norm-matrix-bound*:
fixes *A*::(*'a::real-normed-algebra-1*) ^*'n*^*'m*
shows $\|x\| = 1 \Longrightarrow \|A *v\ x\| \leq \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
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 $\|(\sum i \in UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \leq (\sum i \in UNIV.\ \|A\ \$\ j\ \$\ i * x\ \$\ i\|)$
using *norm-sum* **by** *blast*
also have ... $\leq (\sum i \in UNIV.\ (\|A\ \$\ j\ \$\ i\|) * (\|x\ \$\ i\|))$
by (*simp add*: *norm-mult-ineq sum-mono*)
also have ... $\leq (\sum i \in UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$

using *xi-le1* **by** (*simp add*: *sum-mono mult-left-le*)
finally have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \le (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$ **by** *simp*}
hence $\bigwedge j.\ \|(A *v\ x)\ \$\ j\| \le ((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j$
unfolding *matrix-vector-mult-def* **by** *simp*
hence $(\sum j{\in}UNIV.\ (\|(A *v\ x)\ \$\ j\|)^2) \le (\sum j{\in}UNIV.\ (\|((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j\|)^2)$
by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)
thus $\|A *v\ x\| \le \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
unfolding *norm-vec-def L2-set-def* **by** *simp*
qed

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

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

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

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

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

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

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

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

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

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

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

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

lemma *op-norm-matrix-matrix-mult-le*:
fixes *A*::(′*a*::*real-normed-algebra-1*) ^′*n* ^′*m*
shows $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$
proof(*rule onorm-le*)
have $0 \leq (\|A\|_{op})$
by(*rule onorm-pos-le*[*OF blin-norm-matrix*])
fix *x* **have** $\|A ** B *v\ x\| = \|A *v\ (B *v\ x)\|$
by (*simp add*: *matrix-vector-mul-assoc*)
also have ... $\leq (\|A\|_{op}) * (\|B *v\ x\|)$
by (*simp add*: *norm-matrix-le-mult-op-norm*[*of - B *v x*])
also have ... $\leq (\|A\|_{op}) * ((\|B\|_{op}) * (\|x\|))$
using *norm-matrix-le-mult-op-norm*[*of B x*] ‹$0 \leq (\|A\|_{op})$› *mult-left-mono* **by**
blast
finally show $\|A ** B *v\ x\| \leq (\|A\|_{op}) * (\|B\|_{op}) * (\|x\|)$
by *simp*

qed

lemma *norm-matrix-vec-mult-le-transpose*:
$\|x\| = 1 \Longrightarrow (\|A *v\ x\|) \leq sqrt\ (\|transpose\ A ** A\|_{op}) * (\|x\|)$ **for** A::*real*^'*n*^'*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*^'*n*^'*m*

proof(*unfold op-norm-def*, *rule cSup-least*[*OF op-norm-set-proptys*(*3*)], *clarsimp*)
fix x::*real*^'*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*^'*n*^'*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 $((\|A\|_{op}))^2 \leq (\|transpose\ A ** A\|_{op})$
using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *op-norm-ge-0* **by** *force*
also have ... $\leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$

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

1.2.2 Matrix maximum norm

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

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

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

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

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

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

1.3 Picard Lindeloef for linear systems

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

lemma *matrix-lipschitz-constant*:
fixes A::*real*$\hat{\ }'n\hat{\ }'n$
shows *dist* $(A *v\ x)$ $(A *v\ y) \leq$ (*real CARD*$('n))^2 * (\|A\|_{max}) *$ *dist x y*
unfolding *dist-norm matrix-vector-mult-diff-distrib*[*symmetric*]
proof(*subst mult-norm-matrix-sgn-eq*[*symmetric*])
have $\|A\|_{op} \leq (\|A\|_{max}) *$ (*real CARD*$('n)$ $*$ *real CARD*$('n)$)
by (*metis* (*no-types*) *Groups.mult-ac*(*2*) *op-norm-le-max-norm*)
then have $(\|A\|_{op}) * (\|x - y\|) \leq$ (*real CARD*$('n))^2 * (\|A\|_{max}) * (\|x - y\|)$
by (*metis* (*no-types*, *lifting*) *mult.commute mult-right-mono norm-ge-zero power2-eq-square*)
also have $(\|A *v$ *sgn* $(x - y)\|) * (\|x - y\|) \leq (\|A\|_{op}) * (\|x - y\|)$
by (*simp add*: *norm-sgn-le-op-norm mult-mono′*)
ultimately show $(\|A *v$ *sgn* $(x - y)\|) * (\|x - y\|) \leq$ (*real CARD*$('n))^2 *$
$(\|A\|_{max}) * (\|x - y\|)$
using *order-trans-rules*(*23*) **by** *blast*
qed

lemma *picard-lindeloef-linear-system*:
fixes A::*real*$\hat{\ }'n\hat{\ }'n$
defines $L \equiv$ (*real CARD*$('n))^2 * (\|A\|_{max})$
shows *picard-lindeloef* (λ *t s*. $A *v\ s$) *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=L* **in** *exI*, *safe*)
using *max-norm-ge-0*[*of A*] **unfolding** *assms* **by** *force* (*rule matrix-lipschitz-constant*)

lemma *picard-lindeloef-affine-system*:
fixes A::*real*$\hat{\ }'n\hat{\ }'n$
shows *picard-lindeloef* (λ *t s*. $A *v\ s + b$) *UNIV UNIV 0*
apply(*rule picard-lindeloef-add*[*OF picard-lindeloef-linear-system*])
using *picard-lindeloef-constant* **by** *auto*

1.4 Matrix Exponential

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

1.4.1 Squared matrices operations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.4.2 Squared matrices form Banach space

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

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

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

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

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

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

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

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

end

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

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

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

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

lemma *mtx-vec-prod-minus-ldistrib*: A $*_V$ (c − d) = A $*_V$ c − A $*_V$ d
by (*metis* (*no-types*, *lifting*) *add-diff-cancel diff-add-cancel*
matrix-vector-right-distrib sq-mtx-vec-prod.rep-eq)

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

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

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

definition *norm-sq-mtx* :: $'a$ *sq-mtx* $\Rightarrow$ *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* = $(INF\ e{:}\{0{<}..\}.\ principal\ \{(x, y).\ dist\ x\ y < e\})$

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-zero-iff*)
by(*transfer*, *simp add*: *norm-sq-mtx-def op-norm-scaleR algebra-simps*)+

end

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

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

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

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

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

lemma *norm-eq-norm-transpose*[*simp*]: $\|A^\dagger\| = \|A\|$

using *norm-le-transpose*[*of A*] **and** *norm-le-transpose*[*of* $A^\dagger$] **by** *simp*

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

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

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

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

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

lemma *sq-mtx-norm-times*: $\|A * B\| \leq (\|A\|) * (\|B\|)$ **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*: *sq-mtx-chi-inject vec-eq-iff one-sq-mtx-def zero-sq-mtx-def mat-def*)
by(*transfer*, *simp add*: *scalar-matrix-assoc matrix-scalar-ac*)+

end

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

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

    by (*simp add*: *mtx-vec-prod-minus-rdistrib mtx-vec-prod-canon*)
  ultimately have $\|\mathrm{col}\ i\ (X\ m) - \mathrm{col}\ i\ (X\ n)\| < \varepsilon$
    by *linarith*}
 thus $\exists\, M.\ \forall\, m{\geq}M.\ \forall\, n{\geq}M.\ \|\mathrm{col}\ i\ (X\ m) - \mathrm{col}\ i\ (X\ n)\| < \varepsilon$
  by *blast*
qed

lemma *col-convergent*:
 assumes $\forall\, i.\ (\lambda n.\ \mathrm{col}\ i\ (X\ n)) \longrightarrow L\ \$\ i$
 shows *convergent X*
 unfolding *convergent-def* **proof**(*rule-tac x=sq-mtx-chi* (*transpose L*) **in** *exI*)
 let *?L* = *sq-mtx-chi* (*transpose L*)
 show $X \longrightarrow$ *?L*
 proof(*unfold LIMSEQ-def dist-norm*, *clarsimp*)
  fix ε::*real* **assume** $\varepsilon > 0$
  let *?a* = *CARD*(*'a*) **fix** ε::*real* **assume** $\varepsilon > 0$
  hence ε / *?a* > *0*
   by *simp*
  from *this* **and** *assms* **have** $\forall\, i.\ \exists\ N.\ \forall\, n{\geq}N.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$
   unfolding *LIMSEQ-def dist-norm convergent-def* **by** *blast*
  then obtain *N* **where** $\forall\, i.\ \forall\, n{\geq}N.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$
   using *finite-nat-minimal-witness*[*of* $\lambda\ i\ n.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$] **by** *blast*
  also have $\bigwedge i\ n.\ (\mathrm{col}\ i\ (X\ n) - L\ \$\ i) = (\mathrm{col}\ i\ (X\ n - ?L))$
   unfolding *minus-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
qed

instance *sq-mtx* :: (*finite*) *banach*
proof(*standard*)

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

1.5 Flow for squared matrix systems

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

lemma *mtx-vec-prod-has-derivative-mtx-vec-prod*:
assumes $\bigwedge$ i j. D (λt. (A t) \$\$ i \$ j) $\mapsto$ ($\lambda\tau$. $\tau *_R (A'\ t)$ \$\$ i \$ j) (*at t within s*)
and ($\lambda\tau$. $\tau *_R (A'\ t) *_V x$) = g'
shows D (λt. A t $*_V$ x) $\mapsto$ g' *at t within s*
using *assms*(*2*) **unfolding** *sq-mtx-vec-mult-sum-cols* **apply** *safe*
apply(*rule-tac* $f'1{=}\lambda i\ \tau.\ \tau *_R$ (x \$ i $*_R$ col i (A' t)) **in** *derivative-eq-intros*(*9*))
apply(*simp-all add*: *scaleR-right.sum*)
apply(*rule-tac* $g'1{=}\lambda\tau.\ \tau *_R$ 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 *has-derivative-mtx-ith*:
assumes D A $\mapsto$ (λh. h $*_R$ A' x) *at x within s*
shows D (λt. A t \$\$ i) $\mapsto$ (λh. h $*_R$ A' x \$\$ i) *at x within s*
unfolding *has-derivative-def tendsto-iff dist-norm* **apply** *safe*
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
proof(*clarsimp*)
fix ε::*real* **assume** $0 < \varepsilon$
let *?x* = *netlimit* (*at x within s*) **let** *?*Δ y = y − *?x* **and** *?*ΔA y = A y − A *?x*
let *?P* e = λy. *inverse* |*?*Δ y| $*$ ($\|$*?*ΔA y − *?*Δ y $*_R$ A' $x\|$) < e
let *?Q* = λy. *inverse* |*?*Δ y| $*$ ($\|A$ y \$\$ i − A *?x* \$\$ i − *?*Δ y $*_R$ A' x \$\$ $i\|$) < ε
from *assms* **have** $\forall e{>}0$. *eventually* (*?P* e) (*at x within s*)
unfolding *has-derivative-def tendsto-iff* **by** *auto*
hence *eventually* (*?P* ε) (*at x within s*)
using ‹$0 < \varepsilon$› **by** *blast*

thus $\mathit{eventually}\ ?Q\ (\mathit{at}\ x\ \mathit{within}\ s)$
proof$(\mathit{rule\text{-}tac}\ P{=}?P\ \varepsilon$ **in** $\mathit{eventually\text{-}mono},\ \mathit{simp\text{-}all})$
let $?u\ y\ i = A\ y\ \$\$\ i - A\ ?x\ \$\$\ i - ?\Delta\ y *_R A'\ x\ \$\$\ i$
fix y **assume** hyp: $\mathit{inverse}\ |?\Delta\ y| * (\|?\Delta A\ y - ?\Delta\ y *_R A'\ x\|) < \varepsilon$
have $\|?u\ y\ i\| = \|(?\Delta A\ y - ?\Delta\ y *_R A'\ x)\ \$\$\ i\|$
by *simp*
also have ... $\le (\|?\Delta A\ y - ?\Delta\ y *_R A'\ x\|)$
using *norm-column-le-norm* **by** *blast*
ultimately have $\|?u\ y\ i\| \le \|?\Delta A\ y - ?\Delta\ y *_R A'\ x\|$
by *linarith*
hence $\mathit{inverse}\ |?\Delta\ y| * (\|?u\ y\ i\|) \le \mathit{inverse}\ |?\Delta\ y| * (\|?\Delta A\ y - ?\Delta\ y *_R A'\ x\|)$
by $(\mathit{simp\ add}: \mathit{mult\text{-}left\text{-}mono})$
thus $\mathit{inverse}\ |?\Delta\ y| * (\|?u\ y\ i\|) < \varepsilon$
using *hyp* **by** *linarith*
qed
qed

lemma *exp-has-vderiv-on-linear*:
fixes $A{::}(('a{::}\mathit{finite})\ \mathit{sq\text{-}mtx})$
shows $D\ (\lambda t.\ \mathit{exp}\ ((t - t0) *_R A) *_V x0) = (\lambda t.\ A *_V (\mathit{exp}\ ((t - t0) *_R A) *_V x0))\ \mathit{on}\ T$
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
apply$(\mathit{rule\text{-}tac}\ A'{=}\lambda t.\ A * \mathit{exp}\ ((t - t0) *_R A)$ **in** *mtx-vec-prod-has-derivative-mtx-vec-prod*$)$
apply$(\mathit{rule}\ \mathit{has\text{-}derivative\text{-}vec\text{-}nth})$
apply$(\mathit{rule}\ \mathit{has\text{-}derivative\text{-}mtx\text{-}ith})$
apply$(\mathit{rule\text{-}tac}\ f'{=}\mathit{id}$ **in** *exp-scaleR-has-derivative-right*$)$
apply$(\mathit{rule\text{-}tac}\ f'1{=}\mathit{id}$ **and** $g'1{=}\lambda x.\ 0$ **in** $\mathit{derivative\text{-}eq\text{-}intros}(11))$
apply$(\mathit{rule}\ \mathit{derivative\text{-}eq\text{-}intros})$
by$(\mathit{simp\text{-}all\ add}$: *fun-eq-iff exp-times-scaleR-commute sq-mtx-times-vec-assoc*$)$

lemma *picard-lindeloef-sq-mtx*:
fixes $A{::}('n{::}\mathit{finite})\ \mathit{sq\text{-}mtx}$
defines $L \equiv (\mathit{real}\ \mathit{CARD}('n))^2 * (\|\mathit{to\text{-}vec}\ A\|_{max})$
shows $\mathit{picard\text{-}lindeloef}\ (\lambda\ t\ s.\ A *_V s)\ \mathit{UNIV}\ \mathit{UNIV}\ t_0$
apply$(\mathit{unfold\text{-}locales},\ \mathit{simp\text{-}all\ add}$: *local-lipschitz-def lipschitz-on-def*, *clarsimp*$)$
apply$(\mathit{rule\text{-}tac}\ x{=}1$ **in** $\mathit{exI},\ \mathit{clarsimp},\ \mathit{rule\text{-}tac}\ x{=}L$ **in** $\mathit{exI},\ \mathit{safe})$
using $\mathit{max\text{-}norm\text{-}ge\text{-}0}[\mathit{of}\ \mathit{to\text{-}vec}\ A]$ **unfolding** *assms* **apply** *force*
by *transfer* $(\mathit{rule}\ \mathit{matrix\text{-}lipschitz\text{-}constant})$

lemma *picard-lindeloef-sq-mtx-affine*:
fixes $A{::}('n{::}\mathit{finite})\ \mathit{sq\text{-}mtx}$
shows $\mathit{picard\text{-}lindeloef}\ (\lambda\ t\ s.\ A *_V s + b)\ \mathit{UNIV}\ \mathit{UNIV}\ t_0$
apply$(\mathit{rule}\ \mathit{picard\text{-}lindeloef\text{-}add}[\mathit{OF}\ \mathit{picard\text{-}lindeloef\text{-}sq\text{-}mtx}])$
using *picard-lindeloef-constant* **by** *auto*

lemma *local-flow-exp*:
fixes $A{::}('n{::}\mathit{finite})\ \mathit{sq\text{-}mtx}$
shows $\mathit{local\text{-}flow}\ ((*_V)\ A)\ \mathit{UNIV}\ \mathit{UNIV}\ (\lambda t\ s.\ \mathit{exp}\ (t *_R A) *_V s)$

unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-sq-mtx* **apply** *blast*
using *exp-has-vderiv-on-linear*[*of 0*] **by** *auto*

end

1.6 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 $skip \equiv (\lambda s.\ \{s\})$

1.6.1 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* = (χ *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*) = (λ*s*. {*vec-upd s x* (*e s*)})

lemma *fbox-assign*[*simp*]: |*x* ::= *e*] *Q* = (λ*s*. *Q* (χ *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 $P \le$ |*G*] *R* $R \le$ |*F*] *Q*
shows $P \le$ |*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 $\equiv$ (λ*s*. *if P s then X s else Y s*)

lemma *fbox-if-then-else*[*simp*]:
|*IF T THEN X ELSE Y*] *Q* = (λ*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 (λ*s*. *P s* $\wedge$ *T s*) $\le$ |*X*] *Q*
and (λ*s*. *P s* $\wedge$ $\neg$ *T s*) $\le$ |*Y*] *Q*
shows $P \le$ |*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* :: $('a \Rightarrow 'a\ set) \Rightarrow nat \Rightarrow ('a \Rightarrow 'a\ set)$
where *kpower f n* = (λ*s*. ((;) *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* :: ($'a \Rightarrow 'a$ *set*) $\Rightarrow$ ($'a \Rightarrow 'a$ *set*) ((-*) [*1000*] *999*)
  where $(f^*)\ s = \bigcup$ {*kpower f n s* |*n*. $n \in$ *UNIV*}

lemma *kpower-inv*:
  fixes $F :: 'a \Rightarrow 'a$ *set*
  assumes $\forall s.\ I\ s \longrightarrow (\forall s'.\ s' \in F\ s \longrightarrow I\ s')$
  shows $\forall s.\ I\ s \longrightarrow (\forall s'.\ s' \in (kpower\ F\ n\ s) \longrightarrow I\ s')$
  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*: $I \leq |F]\ I \Longrightarrow I \leq |F^*]\ I$
  unfolding *kleene-star-def fbox-def* **apply** *clarsimp*
  apply(*unfold le-fun-def*, *subgoal-tac* $\forall x.\ I\ x \longrightarrow (\forall s'.\ s' \in F\ x \longrightarrow I\ s')$)
  using *kpower-inv*[*of I F*] **by** *blast simp*

lemma *fbox-kstarI*:
  assumes $P \leq I$ **and** $I \leq Q$ **and** $I \leq |F]\ I$
  shows $P \leq |F^*]\ Q$
proof−
  have $I \leq |F^*]\ I$
    using *assms*(*3*) *kstar-inv* **by** *blast*
  hence $P \leq |F^*]\ I$
    using *assms*(*1*) **by** *auto*
  also have $|F^*]\ I \leq |F^*]\ Q$
    by (*rule fbox-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* $\equiv (F^*)$

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*

1.6.2 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´=- & - on - - @ - DINV -*))
where $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)$

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*: $P \leq I \Longrightarrow I \leq |x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ I \Longrightarrow (\lambda s.\ I\ s \wedge G\ s) \leq Q \Longrightarrow$
$P \leq |x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I]\ Q$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $b{=}|x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ I$ **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*)

1.6.3 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 $|x' = (\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0]\ Q =$
  $(\lambda s.\ \forall t \in T.\ (\mathcal{P}\ (\lambda\tau.\ s + \tau *_R c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (s + t *_R c))$
  apply(*subst local-flow.fbox-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 $\forall s.\ P\ s \longrightarrow (\forall\ t \in T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (\varphi\ t\ s))$
  shows $P \leq |x' = f\ \&\ G\ on\ T\ UNIV\ @\ 0]\ Q$
  using *assms* **by**(*subst local-flow.fbox-g-ode*) *auto*

lemma *diff-weak-axiom*: $|x' = f\ \&\ G\ on\ T\ S\ @\ t_0]\ Q = |x' = f\ \&\ G\ on\ T\ S\ @\ t_0]\ (\lambda s.\ G\ s \longrightarrow Q\ s)$
  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 $|x' = f\ \&\ G\ on\ T\ S\ @\ t_0]\ C = (\lambda s.\ True)$
    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 $\forall s'.\ s' \in (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ s \longrightarrow C\ s'$
    using *assms*(*1*) **unfolding** *fbox-def* **by** *meson*
  ultimately show $C\ (x\ \tau)$ **by** *blast*
qed

lemma *diff-cut-axiom*:
  assumes *Thyp*: *is-interval* T $t_0 \in T$
    and $|x' = f\ \&\ G\ on\ T\ S\ @\ t_0]\ C = (\lambda s.\ True)$
  shows $|x' = f\ \&\ G\ on\ T\ S\ @\ t_0]\ Q = |x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0]\ Q$
proof(*rule-tac* $f = \lambda\ x.\ |x]\ Q$ **in** *HOL.arg-cong*, *rule ext*, *rule subset-antisym*)
  fix s
  {**fix** s' **assume** $s' \in (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ s$
    then obtain τ::*real* **and** X **where** *x-ivp*: $X \in Sols\ (\lambda t.\ f)\ T\ S\ t_0\ s$

```
      and X 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 t₀ 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 @ t₀) s
      using g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))
    hence ∀ t∈(down T τ). C (X t)
      using assms(3) unfolding fbox-def by meson
    hence s' ∈ (x´= f & (λs. G s ∧ C s) on T S @ t₀) s
      using g-orbitalI[OF x-ivp ‹τ ∈ T›] guard-x ‹X τ = s'› by fastforce}
  thus (x´= f & G on T S @ t₀) s ⊆ (x´= f & (λs. G s ∧ C s) on T S @ t₀) s
    by blast
next show ⋀s. (x´= f & (λs. G s ∧ C s) on T S @ t₀) s ⊆ (x´= f & G on T
S @ t₀) s
    by (auto simp: g-orbital-eq)
qed

lemma diff-cut-rule:
  assumes Thyp: is-interval T t₀ ∈ T
    and fbox-C: P ≤ |x´= f & G on T S @ t₀] C
    and fbox-Q: P ≤ |x´= f & (λs. G s ∧ C s) on T S @ t₀] Q
  shows P ≤ |x´= f & G on T S @ t₀] Q
proof(subst fbox-def, subst g-orbital-eq, clarsimp)
  fix t::real and X::real ⇒ 'a and s assume P s and t ∈ T
    and x-ivp:X ∈ Sols (λt. f) T S t₀ s
    and guard-x:∀τ. τ ∈ T ∧ τ ≤ t ⟶ G (X τ)
  have ∀τ∈(down T t). X τ ∈ (x´= f & G on T S @ t₀) s
    using g-orbitalI[OF x-ivp] guard-x by auto
  hence ∀τ∈(down T t). C (X τ)
    using fbox-C ‹P s› by (subst (asm) fbox-def, auto)
  hence X t ∈ (x´= f & (λs. G s ∧ C s) on T S @ t₀) s
    using guard-x ‹t ∈ 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)⇒'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)⇒'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 $\forall s.\ P\ s \longrightarrow (\forall t.\ (\forall \tau{\leq}t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $P \leq |x\acute{}= f\ \&\ G]\ Q$
apply(*rule diff-solve-rule*[*OF assms*(*1*)])
using *assms*(*2*) **by** *simp*

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

lemma *DW*: $|x\acute{}= f\ \&\ G]\ Q = |x\acute{}= f\ \&\ G]\ (\lambda s.\ G\ s \longrightarrow Q\ s)$
by (*rule diff-weak-axiom*)

lemma *dW*: $G \leq Q \Longrightarrow P \leq |x\acute{}= f\ \&\ G]\ Q$
by (*rule diff-weak-rule*)

lemma *DC*:
assumes $|x\acute{}= f\ \&\ G]\ C = (\lambda s.\ True)$
shows $|x\acute{}= f\ \&\ G]\ Q = |x\acute{}= f\ \&\ (\lambda s.\ G\ s \wedge C\ s)]\ Q$
by (*rule diff-cut-axiom*) (*auto simp*: *assms*)

lemma *dC*:
assumes $P \leq |x\acute{}= f\ \&\ G]\ C$
and $P \leq |x\acute{}= f\ \&\ (\lambda s.\ G\ s \wedge C\ s)]\ Q$
shows $P \leq |x\acute{}= f\ \&\ G]\ Q$
apply(*rule diff-cut-rule*)
using *assms* **by** *auto*

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

end

1.6.4 Examples

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

theory *hs-vc-examples*
imports *hs-prelims-matrices hs-vc-spartan*

begin

Preliminary preparation for the examples.

— Finite set of program variables.

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

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

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

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

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

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

abbreviation *val-p* :: *real*^*program-vars* $\Rightarrow$ *string* $\Rightarrow$ *real* (**infixl** $\downharpoonright_V$ *90*)
where *store*$\downharpoonright_V$*var* $\equiv$ *store*\$$\restriction_V$*var*

— Alternative to the finite set of program variables.

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

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

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

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

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

Circular Motion

— Verified with differential invariants.

abbreviation *circular-motion-vec-field* :: *real^program-vars* ⇒ *real^program-vars* (C)
where *circular-motion-vec-field* $s \equiv (\chi\ i.\ \mathit{if}\ i{=}\upharpoonleft_V ''x''\ \mathit{then}\ s\downharpoonright_V ''y''\ \mathit{else}\ -s\downharpoonright_V ''x'')$

lemma *circular-motion-invariants*:
$(\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2) \leq |x\acute{}=C\ \&\ G]\ (\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2)$
by (*auto intro*!: *diff-invariant-rules poly-derivatives simp*: *to-var-inject*)

— Verified with the flow.

abbreviation *circular-motion-flow* :: *real* ⇒ *real^program-vars* ⇒ *real^program-vars* (φ_C)
where $\varphi_C\ t\ s \equiv (\chi\ i.\ \mathit{if}\ i{=}\upharpoonleft_V ''x''\ \mathit{then}\ s\downharpoonright_V ''x'' * \cos t + s\downharpoonright_V ''y'' * \sin t$
$\mathit{else} - s\downharpoonright_V ''x'' * \sin t + s\downharpoonright_V ''y'' * \cos t)$

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV* φ_C
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 program-vars-univ-eq to-var-inject power2-commute*)
apply(*clarsimp*, *case-tac* $i = \upharpoonleft_V ''x''$)
using *program-vars-exhaust* **by** (*force intro*!: *poly-derivatives simp*: *to-var-inject*)+

lemma *circular-motion*:
$(\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2) \leq |x\acute{}=C\ \&\ G]\ (\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2)$
by (*force simp*: *local-flow.fbox-g-ode*[*OF local-flow-circ-motion*] *to-var-inject*)

— Verified by providing dynamics.

lemma *circular-motion-dyn*:
$(\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2) \leq |EVOL\ \varphi_C\ G\ T]\ (\lambda s.\ r^2 = (s\downharpoonright_V ''x'')^2 + (s\downharpoonright_V ''y'')^2)$
by (*force simp*: *to-var-inject*)

no-notation *circular-motion-vec-field* (C)
and *circular-motion-flow* (φ_C)

— Verified as a linear system (using uniqueness).

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

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

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

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

no-notation *circular-motion-sq-mtx* (C)
    and *circular-motion-mtx-flow* (φ_C)

Bouncing Ball

— Verified with differential invariants.

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

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

abbreviation *cnst-acc-vec-field* :: *real* ⇒ *real*ˆ*program-vars* ⇒ *real*ˆ*program-vars* (K)
  where $K\ a\ s \equiv (\chi\ i.\ if\ i{=}(\upharpoonleft_V{''x''})\ then\ s\downharpoonleft_V{''y''}\ else\ a)$

lemma *bouncing-ball-invariants*:
  shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
  $(\lambda s.\ s\downharpoonleft_V{''x''} = h \wedge s\downharpoonleft_V{''y''} = 0) \leq fbox$
  $(LOOP$
    $((x\acute{}=K\ g\ \&\ (\lambda\ s.\ s\downharpoonleft_V{''x''} \geq 0)\ DINV\ (\lambda s.\ 2 * g * s\downharpoonleft_V{''x''} - 2 * g * h - (s\downharpoonleft_V{''y''} * s\downharpoonleft_V{''y''}) = 0))\ ;$
    $(IF\ (\lambda s.\ s\downharpoonleft_V{''x''} = 0)\ THEN\ (\upharpoonleft_V{''y''} ::= (\lambda s.\ - s\downharpoonleft_V{''y''}))\ ELSE\ skip))$

INV ($\lambda s.\ s\downharpoonleft_V{''x''} \geq 0 \wedge 2 * g * s\downharpoonleft_V{''x''} - 2 * g * h - (s\downharpoonleft_V{''y''} * s\downharpoonleft_V{''y''}) = 0$))
($\lambda s.\ 0 \leq s\downharpoonleft_V{''x''} \wedge s\downharpoonleft_V{''x''} \leq h$)
apply(*rule fbox-loopI*, *simp-all*)
apply(*force*, *force simp*: *bb-real-arith*)
by (*rule fbox-g-odei*) (*auto intro*!: *poly-derivatives diff-invariant-rules simp*: *to-var-inject*)

— Verified with the flow.

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

abbreviation *cnst-acc-flow* :: *real* ⇒ *real* ⇒ *real^program-vars* ⇒ *real^program-vars* (φ_K)
where φ_K *a t s* ≡ (χ *i. if i*=($\upharpoonleft_V$ ''x'') *then a* ∗ *t* ^ *2/2* + *s* \$ ($\upharpoonleft_V$ ''y'') ∗ *t* + *s* \$ ($\upharpoonleft_V$ ''x'')
else a ∗ *t* + *s* \$ ($\upharpoonleft_V$ ''y''))

lemma *local-flow-cnst-acc*: *local-flow* (*K g*) *UNIV UNIV* (φ_K *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 program-vars-univ-eq to-var-inject*)
apply(*clarsimp*, *case-tac i* = $\upharpoonleft_V$ ''x'')
using *program-vars-exhaust* **by**(*auto intro*!: *poly-derivatives simp*: *to-var-inject vec-eq-iff*)

lemma [*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
and *pos*: $g * \tau^2 \ / \ 2 + v * \tau + (x{::}real) = 0$
shows $2 * g * h + (g * \tau + v) * (g * \tau + v) = 0$
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 $2 * g * h + (g * \tau + v) * (g * \tau + v) = 0$
by (*simp add*: *add.commute distrib-right power2-eq-square*)
qed

lemma [*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 $?lhs = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
  apply(*subst Rat.sign-simps*(*18*))+
  by(*auto simp*: *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*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
 $(\lambda s.\ s\lfloor_V''x'' = h \wedge s\lfloor_V''y'' = 0) \leq$ *fbox*
 (*LOOP*
  $((x' {=} K\ g\ \&\ (\lambda\ s.\ s\lfloor_V''x'' \geq 0))$;
  $(IF\ (\lambda s.\ s\lfloor_V''x'' = 0)\ THEN\ (\upharpoonright_V''y'' ::= (\lambda s.\ -\ s\lfloor_V''y''))\ ELSE\ skip))$
 $INV\ (\lambda s.\ \ s\lfloor_V''x'' \geq 0 \wedge 2 * g * s\lfloor_V''x'' = 2 * g * h + (s\lfloor_V''y'' * s\lfloor_V''y'')))$
 $(\lambda s.\ 0 \leq s\lfloor_V''x'' \wedge s\lfloor_V''x'' \leq h)$
 apply(*rule fbox-loopI*, *simp-all add*: *local-flow.fbox-g-ode*[*OF local-flow-cnst-acc*])
 by (*auto simp*: *bb-real-arith to-var-inject*)

no-notation *cnst-acc-vec-field* (*K*)
  and *cnst-acc-flow* (φ_K)
  and *to-var* ($\upharpoonright_V$)
  and *val-p* (**infixl** $\lfloor_V$ *90*)

— Verified as a linear system (computing exponential).

abbreviation *cnst-acc-sq-mtx* :: *3 sq-mtx* (*K*)
 where $K \equiv$ *sq-mtx-chi* (χ *i*::*3*. *if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

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

times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

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

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

lemma *bouncing-ball-sq-mtx*:
(λ*s*. *0* ≤ *s*\$*0* ∧ *s*\$*0* = *h* ∧ *s*\$*1* = *0* ∧ *0* > *s*\$*2*) ≤ *fbox*
(*LOOP* ((*x*´=($*_V$) *K* & (λ *s*. *s*\$*0* ≥ *0*)) ;
(*IF* (λ *s*. *s*\$*0* = *0*) *THEN* (*1* ::= (λ*s*. − *s*\$*1*)) *ELSE skip*))
INV (λ*s*. *0*≤*s*\$*0* ∧ *0*>*s*\$*2* ∧ *2* ∗ *s*\$*2* ∗ *s*\$*0* = *2* ∗ *s*\$*2* ∗ *h* + (*s*\$*1* ∗ *s*\$*1*)))
(λ*s*. *0* ≤ *s*\$*0* ∧ *s*\$*0* ≤ *h*)
apply(*rule fbox-loopI*[*of* - (λ*s*. *0*≤*s*\$*0* ∧ *0*>*s*\$*2* ∧ *2* ∗ *s*\$*2* ∗ *s*\$*0* = *2* ∗ *s*\$*2* ∗ *h* + (*s*\$*1* ∗ *s*\$*1*))])
apply(*simp-all add*: *local-flow.fbox-g-ode*[*OF local-flow-exp*] *sq-mtx-vec-prod-eq*)
apply(*force*, *force simp*: *bb-real-arith*)
unfolding *UNIV-3* **apply**(*simp add*: *exp-cnst-acc-sq-mtx-simps*, *safe*)
using *bb-real-arith*(*2*)[*of* - - *h*] **apply** (*force simp*: *field-simps*)
subgoal for *s* τ **using** *bb-real-arith*(*3*)[*of s*\$*2*] **by**(*simp add*: *field-simps*)
done

no-notation *cnst-acc-sq-mtx* (*K*)

Thermostat

typedef *thermostat-vars* = {''*t*'',''*T*'',''*on*'',''*TT*''}
morphisms *to-str to-var*
apply(*rule-tac x*=''*t*'' **in** *exI*)
by *simp*

notation *to-var* (↾$_V$)

lemma *number-of-thermostat-vars*: *CARD*(*thermostat-vars*) = *4*
using *type-definition.card type-definition-thermostat-vars* **by** *fastforce*

instance *thermostat-vars*::*finite*
apply(*standard*)
apply(*subst bij-betw-finite*[*of to-str UNIV* {''*t*'',''*T*'',''*on*'',''*TT*''}])

apply(*rule bij-betwI'*)
apply (*simp add*: *to-str-inject*)
using *to-str* **apply** *blast*
apply (*metis to-var-inverse UNIV-I*)
by *simp*

lemma *thermostat-vars-univ-eq*:
$(UNIV::thermostat\text{-}vars\ set) = \{\restriction_V ''t'', \restriction_V ''T'', \restriction_V ''on'', \restriction_V ''TT''\}$
apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *thermostat-vars-exhaust*: $x{=}\restriction_V ''t'' \vee x{=}\restriction_V ''T'' \vee x{=}\restriction_V ''on'' \vee x{=}\restriction_V ''TT''$
using *thermostat-vars-univ-eq* **by** *auto*

lemma *thermostat-vars-sum*:
fixes f :: *thermostat-vars* $\Rightarrow$ (*'a::banach*)
shows $(\sum (i::thermostat\text{-}vars) \in UNIV.\ f\ i) =$
$f\ (\restriction_V ''t'') + f\ (\restriction_V ''T'') + f\ (\restriction_V ''on'') + f\ (\restriction_V ''TT'')$
unfolding *thermostat-vars-univ-eq* **by** (*simp add*: *to-var-inject*)

abbreviation *val-T* :: *real^thermostat-vars* $\Rightarrow$ *string* $\Rightarrow$ *real* (**infixl** $\downharpoonleft_V$ *90*)
where $store \downharpoonleft_V var \equiv store\$\restriction_V var$

lemma *thermostat-vars-allI*:
$P\ (\restriction_V ''t'') \Longrightarrow P\ (\restriction_V ''T'') \Longrightarrow P\ (\restriction_V ''on'') \Longrightarrow P\ (\restriction_V ''TT'') \Longrightarrow \forall i.\ P\ i$
using *thermostat-vars-exhaust* **by** *metis*

abbreviation *temp-vec-field* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real^thermostat-vars* $\Rightarrow$ *real^thermostat-vars* (f_T)
where $f_T\ a\ L\ s \equiv (\chi\ i.\ if\ i{=}\restriction_V ''t''\ then\ 1\ else\ (if\ i{=}\restriction_V ''T''\ then\ -\ a * (s\downharpoonleft_V ''T''$
$-\ L)\ else\ 0))$

abbreviation *temp-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real^thermostat-vars* $\Rightarrow$ *real^thermostat-vars* (φ_T)
where $\varphi_T\ a\ L\ t\ s \equiv (\chi\ i.\ if\ i{=}\restriction_V ''T''\ then\ -\ exp(-a * t) * (L - s\downharpoonleft_V ''T'') +$
$L\ else$
$(if\ i{=}\restriction_V ''t''\ then\ t + s\downharpoonleft_V ''t''\ else$
$(if\ i{=}\restriction_V ''on''\ then\ s\downharpoonleft_V ''on''\ else\ s\downharpoonleft_V ''TT'')))$

lemma *norm-diff-temp-dyn*: $0 < a \Longrightarrow \|f_T\ a\ L\ s_1 - f_T\ a\ L\ s_2\| = |a| * |s_1\downharpoonleft_V ''T''$
$-\ s_2\downharpoonleft_V ''T''|$
proof(*simp add*: *norm-vec-def L2-set-def thermostat-vars-sum to-var-inject*)
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 $|a * (s_1\downharpoonleft_V ''T'' + -\ L) + -\ (a * (s_2\downharpoonleft_V ''T'' + -\ L))| = a * |s_1\downharpoonleft_V ''T'' +$
$-\ s_2\downharpoonleft_V ''T''|$
using *a1* **by** (*simp add*: *abs-mult*)

thus $|a * (s_2 \downharpoonright_V {''T''} - L) - a * (s_1 \downharpoonright_V {''T''} - L)| = a * |s_1 \downharpoonright_V {''T''} - s_2 \downharpoonright_V {''T''}|$
using *f2 minus-real-def* **by** *presburger*
qed

lemma *local-lipschitz-temp-dyn*:
assumes $0 < (a{::}real)$
shows *local-lipschitz UNIV UNIV* $(\lambda t{::}real.\ f_T\ a\ L)$
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*)
apply(*unfold thermostat-vars-univ-eq*, *simp add*: *to-var-inject*, *clarsimp*)
unfolding *real-sqrt-abs*[*symmetric*] **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-temp-up*: $a > 0 \Longrightarrow$ *local-flow* $(f_T\ a\ L)$ *UNIV UNIV* $(\varphi_T\ a\ L)$
apply(*unfold-locales*, *simp-all*)
using *local-lipschitz-temp-dyn* **apply** *blast*
apply(*rule thermostat-vars-allI*, *simp-all add*: *to-var-inject*)
using *thermostat-vars-exhaust* **by** (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff to-var-inject*)

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

lemma *thermostat*:
assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $(\lambda s.\ Tmin \leq s\downharpoonleft_V{}''T'' \wedge s\downharpoonleft_V{}''T'' \leq Tmax \wedge s\downharpoonleft_V{}''on''{=}0) \leq$
$|LOOP$
— control
$((({\upharpoonleft_V}''t'')::=(\lambda s.0));(({\upharpoonleft_V}''TT'')::=(\lambda s.\ s\downharpoonleft_V{}''T''));$
$(IF\ (\lambda s.\ s\downharpoonleft_V{}''on''{=}0 \wedge s\downharpoonleft_V{}''TT''{\leq}Tmin + 1)\ THEN\ ({\upharpoonleft_V}''on'' ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\downharpoonleft_V{}''on''{=}1 \wedge s\downharpoonleft_V{}''TT''{\geq}Tmax - 1)\ THEN\ ({\upharpoonleft_V}''on'' ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\downharpoonleft_V{}''on''{=}0)\ THEN\ (x\acute{}{=}(f_T\ a\ 0)\ \&\ (\lambda s.\ s\downharpoonleft_V{}''t'' \leq -\ (ln\ (Tmin/s\downharpoonleft_V{}''TT''))/a)\ on\ \{0..t\}\ UNIV\ @\ 0)$
$ELSE\ (x\acute{}{=}(f_T\ a\ L)\ \&\ (\lambda s.\ s\downharpoonleft_V{}''t'' \leq -\ (ln\ ((L{-}Tmax)/(L{-}s\downharpoonleft_V{}''TT'')))/a)\ on\ \{0..t\}\ UNIV\ @\ 0))\ )$
$INV\ (\lambda s.\ Tmin \leq s\downharpoonleft_V{}''T'' \wedge s\downharpoonleft_V{}''T''{\leq}Tmax \wedge (s\downharpoonleft_V{}''on''{=}0 \vee s\downharpoonleft_V{}''on''{=}1))]$
$(\lambda s.\ Tmin \leq s\$\upharpoonleft_V{}''T'' \wedge s\$\upharpoonleft_V{}''T'' \leq Tmax)$
apply(*rule fbox-loopI*, *simp-all add*: *wlp-temp-dyn[OF assms(1,2)] le-fun-def to-var-inject*, *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 *thermostat-vars.to-var* ($\upharpoonleft_V$)
and *val-T* (**infixl** $\downharpoonleft_V$ *90*)
and *temp-vec-field* (f_T)
and *temp-flow* (φ_T)

Thermostat

abbreviation *tank-vec-field* :: *real* ⇒ *real^4* ⇒ *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* ⇒ *real* ⇒ *real^4* ⇒ *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* ⇒ *real* ⇒ *real^4* ⇒ *bool* (*G*)
  where $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *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^4* ⇒ *bool* (*dI*)
  where $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k * s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$
    $hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

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 $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) * \tau + y$
    and $y \leq hmax \Longrightarrow y - c_o * \tau \leq hmax$
  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*)

lemma *tank-flow*:
  assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
  shows *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 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)

end

1.7 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* - - @ -))

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

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 $vec\text{-}upd\ s\ i\ a = (\chi\ j.\ (((\$)\ s)(i := a))\ j)$

definition *assign* :: $'n \Rightarrow ('a\hat{\ }'n \Rightarrow 'a) \Rightarrow ('a\hat{\ }'n) \Rightarrow ('a\hat{\ }'n)\ set\ ((2\text{-} ::= \text{-})\ [70, 65]\ 61)$
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 $P \leq fb_{\mathcal{F}}\ F\ R\ R \leq fb_{\mathcal{F}}\ G\ Q$
shows $P \leq fb_{\mathcal{F}}\ (F\ ;\ G)\ Q$
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**
$IF\ P\ THEN\ X\ ELSE\ Y = (\lambda\ x.\ if\ P\ x\ then\ X\ x\ else\ Y\ x)$

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 $P \cap \{s.\ T\ s\} \leq fb_{\mathcal{F}}\ X\ Q$
and $P \cap \{s.\ \neg\ T\ s\} \leq fb_{\mathcal{F}}\ Y\ Q$
shows $P \leq fb_{\mathcal{F}}\ (IF\ T\ THEN\ X\ ELSE\ Y)\ Q$

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*

1.7.2 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*: $fb_{\mathcal{F}}$ $(x' = f$ & 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*: $fb_{\mathcal{F}}$ $(x' = f$ & 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*: $fb_{\mathcal{F}}$ $(x' = f$ & 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*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow fb_{\mathcal{F}}$ $(x'=f$ & 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 $(x' = f$ & G *on* $T\ S$ @ t_0 *DINV* $I) = (x' = f$ & G *on* $T\ S$ @ $t_0)$

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

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

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

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

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

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

context *local-flow*
begin

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

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

end

lemma *ffb-g-odei*: $P \leq \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.$

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*)

1.7.3 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 *fb*$_{\mathcal{F}}$ (*x*´=(λ*s. c*) & *G on T UNIV* @ *0*) *Q* =
{*s.* $\forall$ *t*$\in$*T*. ($\mathcal{P}$ ($\lambda\tau$. *s* + τ $*_R$ *c*) (*down T t*) $\subseteq$ {*s. G s*}) $\longrightarrow$ (*s* + *t* $*_R$ *c*) $\in$ *Q*}
apply(*subst local-flow.ffb-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. s* $\in$ *P* $\longrightarrow$ ($\forall$ *t*$\in$*T*. ($\mathcal{P}$ (λ*t.* φ *t s*) (*down T t*) $\subseteq$ {*s. G s*}) $\longrightarrow$ (φ *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$$\in$(*down T t*). *x* τ $\in$ (*x*´= *f* & *G on T S* @ t_0) *s*
shows $\forall$ $\tau$$\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 $\forall$ *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* τ) **by** *blast*
qed

lemma *diff-cut-axiom*:

assumes *Thyp*: *is-interval* T $t_0 \in T$
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 = $fb_{\mathcal{F}}$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) Q
proof(*rule-tac* $f = \lambda\ x.\ fb_{\mathcal{F}}\ x\ Q$ **in** *HOL.arg-cong*, *rule ext*, *rule subset-antisym*)
fix s
{fix s' **assume** $s' \in$ ($x' = f$ & 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). $X\ t \in$ ($x' = f$ & 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* **unfolding** *ffb-eq* **by** *fastforce*
hence $s' \in$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) s
using *g-orbitalI*[*OF x-ivp* ‹$\tau \in T$›] *guard-x* ‹$X\ \tau = s'$› **by** *fastforce***}**
thus ($x' = f$ & G *on* T S @ t_0) $s \subseteq$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) s
by *blast*
next show $\bigwedge s.$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) $s \subseteq$ ($x' = f$ & G *on* T S @ t_0) s
by (*auto simp*: *g-orbital-eq*)
qed

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and *ffb-C*: $P \leq fb_{\mathcal{F}}$ ($x' = f$ & G *on* T S @ t_0) $\{s.\ C\ s\}$
and *ffb-Q*: $P \leq fb_{\mathcal{F}}$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) Q
shows $P \leq fb_{\mathcal{F}}$ ($x' = f$ & G *on* T S @ t_0) Q
proof(*subst ffb-eq*, *subst g-orbital-eq*, *clarsimp*)
fix t::*real* **and** X::*real* $\Rightarrow$ $'a$ **and** s **assume** $s \in P$ **and** $t \in T$
and *x-ivp*:$X \in$ *Sols* ($\lambda t.\ f$) T S t_0 s
and *guard-x*:$\forall\, \tau.$ *s2p* T $\tau \wedge \tau \leq t \longrightarrow G$ (X τ)
have $\forall\, r \in$(*down* T t). $X\ r \in$ ($x' = f$ & G *on* 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 *ffb-C* ‹$s \in P$› **by** (*subst* (*asm*) *ffb-eq*, *auto*)
hence $X\ t \in$ ($x' = f$ & ($\lambda s.\ G\ s \wedge C\ s$) *on* T S @ t_0) s
using *guard-x* ‹$t \in T$› **by** (*auto intro*!: *g-orbitalI x-ivp*)
thus (X t) $\in Q$
using ‹$s \in P$› *ffb-Q* **by** (*subst* (*asm*) *ffb-eq*) *auto*
qed

The rules of dL

abbreviation *g-global-orbit* ::(($'a$::*banach*)$\Rightarrow$$'a$) $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *set*
((*1x´=- & -*)) **where** ($x' = f$ & G) $\equiv$ ($x' = f$ & 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 $\forall s.\ s \in P \longrightarrow (\forall t.\ (\forall \tau{\le}t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)$
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

1.7.4 Examples

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

theory *cat2funcset-examples*

imports *../hs-prelims-matrices 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* :: $real\hat{}2 \Rightarrow real\hat{}2$ (f)
where $f\ s \equiv (\chi\ i.\ if\ i{=}0\ then\ s\$1\ else\ -s\$0)$

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

— Verified with the flow.

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

— Verified by providing the dynamics

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

— Verified as a linear system (using uniqueness).

abbreviation *pend-sq-mtx* :: *2 sq-mtx* (A)
where $A \equiv$ *sq-mtx-chi* (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-exp*, *symmetric*])
apply(*rule ivp-solsI*, *clarsimp*)
unfolding *sq-mtx-vec-prod-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *pendulum-sq-mtx*: $\{s.\ r^2 = (s\$0)^2 + (s\$1)^2\} \le fb_{\mathcal{F}}\ (x' = (*_V)\ A\ \&\ G)\ \{s.\ r^2 = (s\$0)^2 + (s\$1)^2\}$
unfolding *local-flow.ffb-g-ode*[*OF local-flow-exp*] *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 $(x{::}real) \le h$
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* :: $real \Rightarrow real\hat{}2 \Rightarrow real\hat{}2$ (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* (
$(x\acute{}{=}(f\ g)\ \&\ (\lambda\ s.\ s\$0 \geq 0)\ DINV\ (\lambda s.\ 2 \cdot g \cdot s\$0 - 2 \cdot g \cdot h - s\$1 \cdot s\$1 = 0))$;
$(IF\ (\lambda\ s.\ s\$0 = 0)\ THEN\ (1 ::= (\lambda s.\ - s\$1))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$0 \wedge 2 \cdot g \cdot s\$0 - 2 \cdot g \cdot h - s\$1 \cdot s\$1 = 0))$
$\{s.\ 0 \leq s\$0 \wedge s\$0 \leq h\}$
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* :: $real \Rightarrow real \Rightarrow real\hat{}2 \Rightarrow real\hat{}2$ (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 $?lhs = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
apply(*subst Rat.sign-simps*(*18*))+
by(*auto simp*: *semiring-normalization-rules*(*29*))
also have $... = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$ (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** $?rhs = g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$
by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
also have ... = *?middle*
by (*simp add*: *semiring-normalization-rules*(*29*))
finally have *?rhs* = *?middle*.}
ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$0 = h \wedge s\$1 = 0\} \leq fb_{\mathcal{F}}$
(*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* ≡ *sq-mtx-chi* (χ *i. if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

lemma *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 sq-mtx-chi-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-exp*] *sq-mtx-vec-prod-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 $f\ a\ L\ s \equiv (\chi\ i.\ if\ i = 1\ then\ 1\ else\ (if\ i = 0\ then\ -\ a * (s\$0 - L)\ else\ 0))$

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

— 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

1.8 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

1.8.1 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* (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 $p \cdot \mathsf{tt}$ r) x $q \Longrightarrow H$ (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 = (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 (tt $p \cdot \mathsf{tt}$ r) x $q \Longrightarrow H$ p (tt $r \cdot x$) q
using *Hoare-def n-de-morgan-var2 phl.ht-at-phl-export1* **by** *auto*

lemma *H-while*: H (tt $p \cdot \mathsf{tt}$ r) x $p \Longrightarrow H$ p (*while r do x od*) (tt $p \cdot n$ r)
proof −
assume *a1*: H (tt $p \cdot \mathsf{tt}$ r) x p
have tt (tt $p \cdot n$ r) = n $r \cdot \mathsf{tt}$ $p \cdot n$ 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 $p \le \mathsf{tt}$ $i \Longrightarrow \mathsf{tt}$ $i \cdot n$ $r \le \mathsf{tt}$ $q \Longrightarrow H$ (tt $i \cdot \mathsf{tt}$ 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 $p \le \mathsf{tt}$ i **and** H i x i **and** (tt i) $\le$ (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*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i \Longrightarrow H\ i\ x\ i \Longrightarrow \mathfrak{tt}\ i \leq \mathfrak{tt}\ q \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ q$
  unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

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

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

end

1.8.2 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*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ p' \Longrightarrow \mathfrak{tt}\ q' \leq \mathfrak{tt}\ q \Longrightarrow Ref\ p'\ q' \leq Ref\ p\ q$
proof –
  assume *h1*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ p'$ **and** *h2*: $\mathfrak{tt}\ q' \leq \mathfrak{tt}\ q$
  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* $(Ref\ (\mathtt{tt}\ v \cdot \mathtt{tt}\ p)\ q)$ *else* $(Ref\ (n\ v \cdot \mathtt{tt}\ p)\ q)$ *fi* $\leq Ref\ p\ q$
proof –
  have $H\ (\mathtt{tt}\ v \cdot \mathtt{tt}\ p)\ (Ref\ (\mathtt{tt}\ v \cdot \mathtt{tt}\ p)\ q)\ q$ **and** $H\ (n\ v \cdot \mathtt{tt}\ p)\ (Ref\ (n\ v \cdot \mathtt{tt}\ p)\ q)\ q$
    by (*simp add*: *R1*)+
  hence $H\ p$ (*if* v *then* $(Ref\ (\mathtt{tt}\ v \cdot \mathtt{tt}\ p)\ q)$ *else* $(Ref\ (n\ v \cdot \mathtt{tt}\ p)\ q)$ *fi*) q
    by (*simp add*: *H-cond n-mult-comm*)
  thus *?thesis*
    by (*rule R2*)
qed

— While loop

lemma *R-while*: *while* q *do* $(Ref\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ q)\ p)$ *od* $\leq$ *Ref* $p\ (\mathfrak{tt}\ p \cdot n\ q)$
proof –
  have $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ q)\ (Ref\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ q)\ p)\ \ p$
    by (*simp-all add*: *R1*)
  hence $H\ p$ (*while* q *do* $(Ref\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ q)\ p)$ *od*) $(\mathfrak{tt}\ p \cdot n\ q)$
    by (*simp add*: *H-while*)
  thus *?thesis*
    by (*rule R2*)
qed

— Finite iteration

lemma *R-star*: $(Ref\ i\ i)^\star \leq Ref\ i\ i$
proof –
  have $H\ i\ (Ref\ i\ i)\ i$
    using *R1* **by** *blast*
  hence $H\ i\ ((Ref\ i\ i)^\star)\ i$
    using *H-star* **by** *blast*
  thus $Ref\ i\ i^\star \leq Ref\ i\ i$
    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*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i \Longrightarrow \mathfrak{tt}\ i \leq \mathfrak{tt}\ q \Longrightarrow Ref\ i\ i \leq Ref\ p\ q$
  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*)

1.8.3 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

1.8.4 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* $(\cup)$ (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**
$rel\text{-}R\ P\ Q = \bigcup\{X.\ rel\text{-}kat.Hoare\ P\ X\ Q\}$

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**
$rel\text{-}ad\ R = \{(x,x) \mid x.\ \neg\ (\exists y.\ (x,y) \in R)\}$

interpretation *rel-aka*: *antidomain-kleene-algebra rel-ad* $(\cup)$ (O) *Id* {} $(\subseteq)$ $(\subset)$

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

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

lemma *nd-fun-n-op-one*[*nd-fun-aka*]: $n\ (n\ (1{::}'a\ \mathit{nd\text{-}fun})) = 1$
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* $P\ Q \equiv (\lambda s.\ \bigcup\{(f_\bullet)\ s | f.\ \mathit{Hoare}\ P\ f\ Q\})^\bullet$

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

1.9 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

1.9.1 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* ⇒ *'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*

abbreviation *loopi-sugar* :: *'a rel* ⇒ *'a pred* ⇒ *'a rel* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP R INV I* ≡ *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*

1.9.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: $(('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow$ *'b pred* ⇒ *'a set* ⇒ *'b rel* (*EVOL*)
where *EVOL* φ *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* ⇒ *real set* ⇒ *'a set* ⇒ *real* ⇒ *'a rel* ((*1x´=- & - on - - @ -*))

where $(x' = f \ \& \ G \ on \ T \ S \ @ \ t_0) = \{(s,s') \mid s \ s'. \ s' \in g\text{-}orbital \ f \ G \ T \ S \ t_0 \ s\}$

lemma *wp-g-orbital*: $wp \ (x' = f \ \& \ G \ on \ T \ S \ @ \ t_0) \ \lceil Q \rceil =$
$\lceil \lambda \ s. \ \forall X \in Sols \ (\lambda t. \ f) \ T \ S \ t_0 \ s. \ \forall t \in T. \ (\forall \tau \in down \ T \ t. \ G \ (X \ \tau)) \longrightarrow Q \ (X \ t) \rceil$
unfolding *g-orbital-eq wp-rel ivp-sols-def g-ode-def* **by** *auto*

context *local-flow*
begin

lemma *wp-g-ode*: $wp \ (x' = f \ \& \ G \ on \ T \ S \ @ \ 0) \ \lceil Q \rceil =$
$\lceil \lambda \ s. \ s \in S \longrightarrow (\forall t \in T. \ (\forall \tau \in down \ T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)) \rceil$
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*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow wp \ (x' = f \ \& \ G \ on \ \{0..t\} \ S \ @ \ 0) \ \lceil Q \rceil$
$=$
$\lceil \lambda s. \ s \in S \longrightarrow (\forall t \in \{0..t\}. \ (\forall \tau \in \{0..t\}. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)) \rceil$
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´=- & - 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 rel-aka.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-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* ($x' = f$ & ($\lambda s.$ *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-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 simp*: *p2r-def*)

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 rel-aka.fbox-iso*, *simp*)

1.9.3 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 $wp\ (x\acute{}{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil =$
  $\lceil \lambda s.\ \forall t{\in}T.\ (\mathcal{P}\ (\lambda t.\ s + t *_R c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (s + t *_R c)\rceil$
  apply(*subst local-flow.wp-g-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 $\lceil P\rceil \le wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil$
  using *assms* **by**(*subst local-flow.wp-g-ode*, *auto*)

lemma *diff-weak-axiom*:
  $wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil = wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil \lambda\ s.\ G\ s \longrightarrow Q\ s\rceil$
  unfolding *wp-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*:
  assumes $\lceil G\rceil \le \lceil Q\rceil$
  shows $\lceil P\rceil \le wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
  using *assms* **apply**(*subst wp-rel*)
  by(*auto simp*: *g-orbital-eq g-ode-def*)

lemma *wp-g-evol-IdD*:
  assumes $wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil = Id$
    and $\forall \tau{\in}(down\ T\ t).\ (s,\ x\ \tau) \in (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)$
  shows $\forall \tau{\in}(down\ T\ t).\ C\ (x\ \tau)$
proof
  fix τ **assume** $\tau \in (down\ T\ t)$
  hence $x\ \tau \in$ *g-orbital* $f\ G\ T\ S\ t_0\ s$
    using *assms*(*2*) **unfolding** *g-ode-def* **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-rel g-ode-def* **by**(*auto simp*: *p2r-def*)
  ultimately show $C\ (x\ \tau)$
    by *blast*
qed

lemma *diff-cut-axiom*:
  assumes *Thyp*: *is-interval* $T\ t_0 \in T$
    and $wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil = Id$
  shows $wp\ (x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil = wp\ (x\acute{}{=}\ f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
proof(*rule-tac* $f{=}\lambda\ x.\ wp\ x\ \lceil Q\rceil$ **in** *HOL.arg-cong*, *rule subset-antisym*)
  show $(x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0) \subseteq (x\acute{}{=}f\ \&\ \lambda s.\ G\ s \wedge C\ s\ on\ T\ S\ @\ t_0)$
  proof(*clarsimp simp*: *g-ode-def*)

```
    fix s and s' assume s' ∈ g-orbital f G T S t0 s
    then obtain τ::real and X where x-ivp: X ∈ Sols (λt. f) T S t0 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 t0 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 by auto
    ultimately have ∀ t∈(down T τ). X t ∈ g-orbital f G T S t0 s
      using g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))
    hence ∀ t∈(down T τ). C (X t)
      using wp-g-evol-IdD[OF assms(3)] unfolding g-ode-def by blast
    thus s' ∈ g-orbital f (λs. G s ∧ C s) T S t0 s
      using g-orbitalI[OF x-ivp ‹τ ∈ T›] guard-x ‹X τ = s'› by fastforce
  qed
next show (x´=f & λs. G s ∧ C s on T S @ t0) ⊆ (x´=f & G on T S @ t0)
    by (auto simp: g-orbital-eq g-ode-def)
qed

lemma diff-cut-rule:
  assumes Thyp: is-interval T t0 ∈ T
    and wp-C: ⌈P⌉ ≤ wp (x´= f & G on T S @ t0) ⌈C⌉
    and wp-Q: ⌈P⌉ ⊆ wp (x´= f & (λs. G s ∧ C s) on T S @ t0) ⌈Q⌉
  shows ⌈P⌉ ⊆ wp (x´= f & G on T S @ t0) ⌈Q⌉
proof(subst wp-rel, simp add: g-orbital-eq p2r-def g-ode-def, clarsimp)
  fix t::real and X::real ⇒ 'a and s assume P s and t ∈ T
    and x-ivp:X ∈ Sols (λt. f) T S t0 s
    and guard-x:∀ x. x ∈ T ∧ x ≤ t ⟶ G (X x)
  have ∀ t∈(down T t). X t ∈ g-orbital f G T S t0 s
    using g-orbitalI[OF x-ivp] guard-x by auto
  hence ∀ t∈(down T t). C (X t)
    using wp-C ‹P s› by (subst (asm) wp-rel, auto simp: g-ode-def)
  hence X t ∈ g-orbital f (λs. G s ∧ C s) T S t0 s
    using guard-x ‹t ∈ T› by (auto intro!: g-orbitalI x-ivp)
  thus Q (X t)
    using ‹P s› wp-Q by (subst (asm) wp-rel) (auto simp: g-ode-def)
qed
```

The rules of dL

```
abbreviation g-global-ode ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a rel ((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 rel
  ((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´=(λs. c) & G) ⌈Q⌉ = ⌈λx. ∀ t. (∀ τ≤t. G (x + τ *R c)) ⟶ Q (x
```

$+\ t *_R\ c)\rceil$
  by (*subst diff-solve-axiom[of UNIV]*) *auto*

lemma *solve*:
  assumes *local-flow f UNIV UNIV* φ
    and $\forall s.\ P\ s \longrightarrow (\forall t.\ (\forall \tau{\le}t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  shows $\lceil P\rceil \le wp\ (x\acute{}= f\ \&\ G)\ \lceil Q\rceil$
  apply(*rule diff-solve-rule[OF assms(1)]*)
  using *assms(2)* **by** *simp*

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

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

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

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

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

end

1.10 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

1.10.1 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 $'a\ pred = 'a \Rightarrow bool$

notation *fbox* (*wp*)

no-notation *bqtran* ($\lfloor$-$\rfloor$)
  and *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
  and *Archimedean-Field.floor* ($\lfloor$-$\rfloor$)
  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* :: $'a\ pred \Rightarrow 'a\ nd\text{-}fun$ (($1\lceil$-$\rceil$))
  where $\lceil Q\rceil \equiv (\lambda\ x{::}'a.\ \{s{::}'a.\ s = x \wedge Q\ s\})^\bullet$

lemma *p2ndf-simps*[*simp*]:
  $\lceil P\rceil \leq \lceil Q\rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
  $(\lceil P\rceil = \lceil Q\rceil) = (\forall s.\ P\ s = Q\ s)$
  $(\lceil P\rceil \cdot \lceil Q\rceil) = \lceil\lambda\ s.\ P\ s \wedge Q\ s\rceil$
  $(\lceil P\rceil + \lceil Q\rceil) = \lceil\lambda\ s.\ P\ s \vee Q\ s\rceil$
  $ad\ \lceil P\rceil = \lceil\lambda s.\ \neg\ P\ s\rceil$
  $d\ \lceil P\rceil = \lceil P\rceil\ \lceil P\rceil \leq \eta^\bullet$
  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* :: $'a\ nd\text{-}fun \Rightarrow 'a \Rightarrow bool$ (($1\lfloor$-$\rfloor$))
  where $\lfloor f\rfloor \equiv (\lambda x.\ x \in Domain\ (\mathcal{R}\ (f_\bullet)))$

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

lemma *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-fun*
  where *skip* ≡ *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* ≡ *aka-cond* $\lceil P\rceil$ *X Y*

abbreviation *loopi-sugar* :: *'a nd-fun* ⇒ *'a pred* ⇒ *'a nd-fun* (*LOOP - INV -* [*64*,*64*] *63*)
  where *LOOP R INV I* ≡ *aka-loop-inv R* ⌈*I*⌉

lemma *wp-loopI*: ⌈*P*⌉ ≤ ⌈*I*⌉ ⟹ ⌈*I*⌉ ≤ ⌈*Q*⌉ ⟹ ⌈*I*⌉ ≤ *wp R* ⌈*I*⌉ ⟹ ⌈*P*⌉ ≤ *wp* (*LOOP R INV I*) ⌈*Q*⌉
  using *fbox-loopi*[*of* ⌈*P*⌉] **by** *auto*

1.10.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: ((*'a*::*ord*) ⇒ *'b* ⇒ *'b*) ⇒ *'b pred* ⇒ *'a set* ⇒ *'b nd-fun* (*EVOL*)
  where *EVOL* φ *G T* = (λ*s*. *g-orbit* (λ*t*. φ *t s*) *G T*)$^\bullet$

lemma *wp-g-dyn*[*simp*]:
  fixes φ :: (*'a*::*preorder*) ⇒ *'b* ⇒ *'b*
  shows *wp* (*EVOL* φ *G T*) ⌈*Q*⌉ = ⌈λ*s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*)⌉
  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*)⇒*'a*) ⇒ *'a pred* ⇒ *real set* ⇒ *'a set* ⇒
  real ⇒ *'a nd-fun* ((*1x´*= - & - *on* - - @ -))
  where (*x´*= *f* & *G on T S* @ t_0) ≡ (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) ⌈*Q*⌉=
  ⌈λ *s*. ∀ *X*∈*ivp-sols* (λ*t*. *f*) *T S* t_0 *s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (*X* τ)) ⟶ *Q* (*X t*)⌉
  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*) ⌈*Q*⌉ =
  ⌈λ *s*. *s* ∈ *S* ⟶ (∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))⌉
  unfolding *wp-g-orbital* **apply**(*clarsimp*, *safe*)
    apply(*erule-tac x*=λ*t*. φ *t s* **in** *ballE*)
  using *in-ivp-sols* **apply**(*force*, *force*, *force simp*: *init-time ivp-sols-def*)
  apply(*subgoal-tac* ∀ τ∈*down T t*. *X* τ = φ τ *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* ≥ *0* ⟹ *t* ∈ *T* ⟹ *wp* (*x´*=*f* & *G on* {*0*..*t*} *S* @ *0*) ⌈*Q*⌉ =
  ⌈λ*s*. *s* ∈ *S* ⟶ (∀ *t*∈{*0*..*t*}. (∀ τ∈{*0*..*t*}. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))⌉
  unfolding *wp-g-orbital* **apply**(*clarsimp*, *safe*)
    apply(*erule-tac x*=λ*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 $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$

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

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

lemma *diff-inv-guard-ignore*:
assumes $\lceil I\rceil \le wp\ (x\acute{}= f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$
shows $\lceil I\rceil \le wp\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$
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-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*)

1.10.3 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 $0 \in T$ **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* φ
and $\forall s.\ P\ s \longrightarrow (\forall\ t \in T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P\rceil \leq wp\ (x' = f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil$
using *assms* **by**(*subst local-flow.wp-g-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* $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = \eta^\bullet$
and $\forall \tau \in (down\ T\ t).\ x\ \tau \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
shows $\forall \tau \in (down\ T\ t).\ C\ (x\ \tau)$
proof
fix τ **assume** $\tau \in (down\ T\ t)$
hence $x\ \tau \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
using *assms(2)* **by** *blast*
also have $\forall y.\ y \in (g\text{-}orbital\ f\ G\ T\ S\ t_0\ s) \longrightarrow C\ y$
using *assms(1)* **unfolding** *wp-nd-fun 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* $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = \eta^\bullet$
shows *wp* $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil =$ *wp* $(x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
proof(*rule-tac* $f = \lambda\ x.\ wp\ x\ \lceil Q \rceil$ **in** *HOL.arg-cong*, *rule nd-fun-ext*, *rule subset-antisym*)
fix s **show** $((x' = f\ \&\ G\ on\ T\ S\ @\ t_0)_\bullet)\ s \subseteq ((x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)_\bullet)\ s$
proof(*clarsimp simp*: *g-ode-def*)
fix s' **assume** $s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s$
and $X\ \tau = s'$ **and** $\tau \in T$ **and** *guard-x*:$(\mathcal{P}\ X\ (down\ T\ \tau) \subseteq \{s.\ G\ s\})$
using *g-orbitalD*[*of* s' $f\ G\ T\ S\ t_0\ s$] **by** *blast*
have $\forall t \in (down\ T\ \tau).\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall t \in (down\ T\ \tau).\ t \in T$
using $\langle \tau \in T \rangle$ *Thyp* **by** *auto*
ultimately have $\forall t \in (down\ T\ \tau).\ X\ t \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence $\forall t \in (down\ T\ \tau).\ C\ (X\ t)$
using *wp-g-orbit-IdD*[*OF assms(3)*] **by** *blast*
thus $s' \in g\text{-}orbital\ f\ (\lambda s.\ G\ s \wedge C\ s)\ T\ S\ t_0\ s$
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** $((x' = f\ \&\ \lambda s.\ G\ s \wedge C\ s\ on\ T\ S\ @\ t_0)_\bullet)\ s \subseteq ((x' = f\ \&\ G\ on\ T\ S\ @\ t_0)_\bullet)\ s$
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*: $\lceil P \rceil \le$ *wp* $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil$
and *wp-Q*: $\lceil P \rceil \le$ *wp* $(x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
shows $\lceil P \rceil \le$ *wp* $(x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$

proof(*simp add*: *wp-nd-fun g-orbital-eq g-ode-def*, *clarsimp*)
  fix t::*real* **and** X::*real* $\Rightarrow$ *'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*) *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*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ *'a nd-fun* ((*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 nd-fun*
  ((*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 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 $\lceil P\rceil \leq$ *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* $(x´= f\ \&\ G)\ \lceil C\rceil = \eta^\bullet$
  shows *wp* $(x´= f\ \&\ G)\ \lceil Q\rceil =$ *wp* $(x´= f\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ \lceil Q\rceil$
  apply (*rule diff-cut-axiom*)
  using *assms* **by** *auto*

lemma *dC*:
  assumes $\lceil P\rceil \leq wp\ (x\acute{} = f\ \&\ G)\ \lceil C\rceil$
    and $\lceil P\rceil \leq wp\ (x\acute{} = f\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ \lceil Q\rceil$
  shows $\lceil P\rceil \leq wp\ (x\acute{} = f\ \&\ G)\ \lceil Q\rceil$
  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 $\lceil P\rceil \leq wp\ (x\acute{} = f\ \&\ G)\ \lceil Q\rceil$
  apply(*rule wp-g-orbital-inv*[*OF assms*(*1*) *- assms*(*3*)])
  unfolding *wp-diff-inv* **using** *assms*(*2*) **.**

end

1.10.4 Examples

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

theory *mka-examples*
  imports *../hs-prelims-matrices 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* :: $real\hat{}2 \Rightarrow real\hat{}2\ (f)$
  where $f\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ -s\$1)$

abbreviation *pend-flow* :: $real \Rightarrow real\hat{}2 \Rightarrow real\hat{}2\ (\varphi)$
  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 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 $A \equiv$ *sq-mtx-chi* (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-exp*, *symmetric*])
apply(*rule ivp-solsI*, *simp add*: *sq-mtx-vec-prod-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-exp*] *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* $\Rightarrow$ *real^2* $\Rightarrow$ *real^2* (*f*)

where *f g s* ≡ (*χ 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*:
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 ∗ 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 $g^2 * \tau^2 + 2 * g * v * \tau + v^2 + 2 * g * h = 0$
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* $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
apply(*subst Rat.sign-simps*(*18*))+
by(*auto simp*: *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* ($\lambda\ s.\ s\$1 = 0$) *THEN* (*2* ::= ($\lambda s.\ -\ s\$2$)) *ELSE skip*))
INV ($\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$))
$\lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
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*
$((x\acute{} = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
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* $\equiv$ *sq-mtx-chi* (χ *i*. *if* $i = 1$ *then* e *2 else if* $i = 2$ *then* e *3 else 0*)

lemma *ball-sq-mtx-pow2*: A^2 = *sq-mtx-chi* (χ *i*. *if* $i = 1$ *then* e *3 else 0*)
unfolding *monoid-mult-class.power2-eq-square times-sq-mtx-def*
by (*simp add*: *sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *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 sq-mtx-chi-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*
    $((x\acute{}=(*_V)A\ \&\ (\lambda\ s.\ s\$1 \geq 0));$
    $(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
  $INV\ (\lambda s.\ 0 \leq s\$1 \wedge 0 > s\$3 \wedge 2 \cdot s\$3 \cdot s\$1 = 2 \cdot s\$3 \cdot h + (s\$2 \cdot s\$2)))$
$\lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
apply(*rule wp-loopI*, *simp-all add*: *local-flow.wp-g-ode*[*OF local-flow-exp*])
 apply(*force simp*: *bb-real-arith*)
apply(*simp add*: *sq-mtx-vec-prod-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* :: $real \Rightarrow real \Rightarrow real\hat{}4 \Rightarrow real\hat{}4$ (*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* :: $real \Rightarrow real \Rightarrow real \Rightarrow real\hat{}4 \Rightarrow real\hat{}4$ (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-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*)

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

lemma *thermostat*:
assumes *a* > *0* **and** *0* ≤ *t* **and** *0* < *Tmin* **and** *Tmax* < *L*
shows ⌈λ*s*. *Tmin* ≤ *s*$*1* ∧ *s*$*1* ≤ *Tmax* ∧ *s*$*4* = *0*⌉ ≤ *wp*
(*LOOP*
  — control
  ((*2* ::= (λ*s*. *0*));(*3* ::= (λ*s*. *s*$*1*));
  (*IF* (λ*s*. *s*$*4* = *0* ∧ *s*$*3* ≤ *Tmin* + *1*) *THEN* (*4* ::= (λ*s*.*1*)) *ELSE*
  (*IF* (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *Tmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*));
  — dynamics
  (*IF* (λ*s*. *s*$*4* = *0*) *THEN* (*x*´=(*f a 0*) & (λ*s*. *s*$*2* ≤ − (*ln* (*Tmin*/*s*$*3*))/*a*) *on* {*0*..*t*} *UNIV* @ *0*)
  ELSE (*x*´=(*f a L*) & (λ*s*. *s*$*2* ≤ − (*ln* ((*L*−*Tmax*)/(*L*−*s*$*3*)))/*a*) *on* {*0*..*t*} *UNIV* @ *0*)))
INV (λ*s*. *Tmin* ≤*s*$*1* ∧ *s*$*1* ≤ *Tmax* ∧ (*s*$*4* = *0* ∨ *s*$*4* = *1*)))
⌈λ*s*. *Tmin* ≤ *s*$*1* ∧ *s*$*1* ≤ *Tmax*⌉
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

1.11 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

1.11.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
  and *Archimedean-Field.floor-ceiling-class.floor* (⌊-⌋)
  and *tau* (τ)
  and *proto-near-quantale-class.bres* (**infixr** → *60*)

notation *Id* (*skip*)

— Canonical lifting from predicates to relations and its simplification rules

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

lemma *p2r-simps*[*simp*]:
 ⌈*P*⌉ ≤ ⌈*Q*⌉ = (∀ *s*. *P s* ⟶ *Q s*)
 (⌈*P*⌉ = ⌈*Q*⌉) = (∀ *s*. *P s* = *Q s*)
 (⌈*P*⌉ ; ⌈*Q*⌉) = ⌈λ *s*. *P s* ∧ *Q s*⌉
 (⌈*P*⌉ ∪ ⌈*Q*⌉) = ⌈λ *s*. *P s* ∨ *Q s*⌉
 rel-tests.t ⌈*P*⌉ = ⌈*P*⌉
 (− *Id*) ∪ ⌈*P*⌉ = − ⌈λ*s*. ¬ *P s*⌉
 Id ∩ (− ⌈*P*⌉) = ⌈λ*s*. ¬ *P s*⌉
 unfolding *p2r-def* **by** *auto*

— Meaning of the relational hoare triple

lemma *rel-kat-H*: *rel-kat.Hoare* ⌈*P*⌉ *X* ⌈*Q*⌉ ⟷ (∀ *s s'*. *P s* ⟶ (*s*,*s'*) ∈ *X* ⟶ *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-kat.Hoare* ⌈*P*⌉ *skip* ⌈*P*⌉
 using *rel-kat.H-skip* **by** *blast*

lemma *sH-skip*[*simp*]: *rel-kat.Hoare* ⌈*P*⌉ *skip* ⌈*Q*⌉ ⟷ ⌈*P*⌉ ≤ ⌈*Q*⌉

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* :: $'b \Rightarrow ('a\hat{}'b \Rightarrow 'a) \Rightarrow ('a\hat{}'b)\ rel\ ((2\text{-} ::= \text{-})\ [70,\ 65]\ 61)$
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* :: $'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel\ (IF\ \text{-}\ THEN\ \text{-}$ $ELSE\ \text{-}\ [64,64]\ 63)$
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* :: $'a\ pred \Rightarrow 'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel\ (WHILE\ \text{-}$ $INV\ \text{-}\ DO\ \text{-}\ [64,64,64]\ 63)$
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*

1.11.2 Verification of hybrid programs

— Verification by providing evolution

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

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 *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 $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
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*: $t \in T$ $\forall x.\ x \in T \wedge x \le t \longrightarrow G\ (X\ x)$ $X \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$
and *main*: $s \in S \longrightarrow (\forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
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* $\langle s \in S\rangle$ *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*: $s \in S$ $P\ s$ $t{\in}T$ $\forall \tau.\ \tau \in T \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)$
and *main*: $\forall s.\ P\ s \longrightarrow (\forall X{\in}Sols\ (\lambda t.\ f)\ T\ S\ 0\ s.\ \forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \le t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
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*: $P\ s$ $X \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$ $t \in T$ $\forall \tau.\ \tau \in T \wedge \tau \le t \longrightarrow G\ (X\ \tau)$
and *main*: $\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau.\ \tau \in T \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
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*: $\forall s.\ P\ s \longrightarrow (\forall X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s.\ \forall t \in \{0..\tau\}.$
$(\forall \tau'.\ 0 \leq \tau' \land \tau' \leq \tau \land \tau' \leq t \longrightarrow G\ (X\ \tau')) \longrightarrow Q\ (X\ t))$
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*: $0 \leq \tau$ $\tau \in T$ $P\ s$ $X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$ $t \in \{0..\tau\}$
$\forall \tau'.\ 0 \leq \tau' \land \tau' \leq \tau \land \tau' \leq t \longrightarrow G\ (X\ \tau')$
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*: $X\ 0 = s$ $\forall \tau \in down\ \{0..\tau\}\ t.\ X \in \{0--\tau\} \rightarrow S$
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*:
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* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q\rceil$ = *rel-kat.Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil R\rceil$
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* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
shows *rel-kat.Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q\rceil$
using *assms*(*1*) **apply**(*rule-tac* $p'=\lceil I\rceil$ **in** *rel-kat.H-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* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil$ = *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* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow$
$\lceil\lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I)$ $\lceil Q\rceil$
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*)

1.11.3 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*: $(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \leq$ *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* $\lceil R\rceil\ \lceil Q\rceil \leq$ *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* $\lceil P\rceil\ \lceil R\rceil$; $(x ::= e) \leq$ *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* $\lceil Q\rceil\ \lceil Q\rceil \leq$ *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* $\lceil Q\rceil\ \lceil(\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j))\rceil$; $(x ::= e) \leq$ *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* $\lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil) \leq$ *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*: $X \leq X' \Longrightarrow Y \leq Y' \Longrightarrow$ (*IF P THEN X ELSE Y*) $\leq$ *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* $\lceil\lambda s.\ P\ s \wedge Q\ s\rceil\ \lceil P\rceil) \leq$ *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*: $X \leq X' \Longrightarrow$ (*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*: $X \leq$ *rel-R* $\lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow$ *LOOP X INV I* $\leq$ *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-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-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-R* $\lceil R\rceil\ \lceil Q\rceil \leq$ *rel-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-R $\lceil P\rceil\ \lceil R\rceil$; $(EVOL\ \varphi\ G\ T) \leq$ *rel-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-R* $\lceil Q\rceil\ \lceil Q\rceil \leq$ *rel-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-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-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*) ≤ *rel-R* ⌈λ*s. s*∈*S* ⟶ (∀ *t*∈*T*. (∀ τ∈*down T t. G* (φ τ *s*)) ⟶ *P* (φ *t s*))⌉ ⌈*P*⌉
unfolding *rel-rkat.spec-def* **by** (*rule H-g-ode*, *simp*)

lemma *R-g-ode-rule*: (∀ *s*∈*S*. *P s* ⟶ (∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))) ⟹
(*x´= f & G on T S @ 0*) ≤ *rel-R* ⌈*P*⌉ ⌈*Q*⌉
unfolding *sH-g-ode*[*symmetric*] **by** (*rule rel-rkat.R2*)

lemma *R-g-odel*: *P* = (λ*s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *R* (φ *t s*)) ⟹
(*x´= f & G on T S @ 0*) ; *rel-R* ⌈*R*⌉ ⌈*Q*⌉ ≤ *rel-R* ⌈*P*⌉ ⌈*Q*⌉
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-g-ode-rule*, *simp-all*)

lemma *R-g-oder*: *R* = (λ*s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*)) ⟹

rel-R ⌈*P*⌉ ⌈*R*⌉; (*x´= f & G on T S @ 0*) ≤ *rel-R* ⌈*P*⌉ ⌈*Q*⌉
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-R* ⌈*Q*⌉ ⌈*Q*⌉ ≤ *rel-R* ⌈λ*s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*)⌉ ⌈*Q*⌉
by (*rule R-g-odel*) *simp*

lemma *rel-R* ⌈*Q*⌉ ⌈λ*s*. ∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*)⌉; (*x´= f & G on T S @ 0*) ≤ *rel-R* ⌈*Q*⌉ ⌈*Q*⌉
by (*rule R-g-oder*) *simp*

lemma *R-g-ode-ivl*:
τ ≥ *0* ⟹ τ ∈ *T* ⟹ (∀ *s*∈*S*. *P s* ⟶ (∀ *t*∈{*0*..τ}. (∀ τ∈{*0*..*t*}. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))) ⟹
(*x´= f & G on* {*0*..τ} *S @ 0*) ≤ *rel-R* ⌈*P*⌉ ⌈*Q*⌉
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* ⟹ ⌈*P*⌉ ≤ ⌈*I*⌉ ⟹ ⌈λ*s*. *I s* ∧ *G s*⌉ ≤ ⌈*Q*⌉ ⟹
(*x´=f & G on T S @* t_0 *DINV I*) ≤ *rel-R* ⌈*P*⌉ ⌈*Q*⌉
unfolding *rel-rkat.spec-def* **by** (*auto simp*: *H-g-ode-inv*)

1.11.4 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* $\lceil P\rceil\ (x\acute{}=(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil$
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 *rel-kat.Hoare* $\lceil P\rceil\ (x\acute{}= f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil$
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* $\lceil P\rceil\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
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* $\lceil P\rceil\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil$
and *wp-Q*:*rel-kat.Hoare* $\lceil P\rceil\ (x\acute{}= f\ \&\ (\lambda\ s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
shows *rel-kat.Hoare* $\lceil P\rceil\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
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 $(x\acute{}= f\ \&\ G) \equiv (x\acute{}= f\ \&\ G\ on\ UNIV\ UNIV\ @\ 0)$

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

end

1.11.5 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* ⌈λ*s*. r^2 = (*s$1*)^2 + (*s$2*)^2⌉ (*EVOL* φ *G T*) ⌈λ*s*. r^2 = (*s$1*)^2 + (*s$2*)^2⌉
  by *simp*

— Verified with differential invariants

lemma *pendulum-inv*: *rel-kat.Hoare*
  ⌈λ*s*. r^2 = (*s$1*)^2 + (*s$2*)^2⌉ (*x´=f* & *G*) ⌈λ*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*: *rel-kat.Hoare*
  ⌈λ*s*. r^2 = (*s$1*)^2 + (*s$2*)^2⌉ (*x´=f* & *G*) ⌈λ*s*. r^2 = (*s$1*)^2 + (*s$2*)^2⌉
  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 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 $f\ g\ s \equiv (\chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ g)$

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*

$((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2 = 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
$)\ \lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
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 $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot h = 0$
using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

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

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

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

lemma *bouncing-ball-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* $\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$ $\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil \geq$
  $(LOOP$
    $((EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T);$
     $(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
   $INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
  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* $\Rightarrow$ *real* $\Rightarrow$ *real^4* $\Rightarrow$ *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 *therm-guard* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real^4* $\Rightarrow$ *bool* (G)
  where $G\ Tmin\ Tmax\ a\ L\ s \equiv (s\$2 \leq - (ln\ ((L-(if\ L=0\ then\ Tmin\ else\ Tmax))/(L-s\$3)))/a)$

abbreviation *therm-loop-inv* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real^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^4* $\Rightarrow$ *real^4* (φ)
  where $\varphi\ a\ L\ \tau\ s \equiv (\chi\ i.\ if\ i = 1\ then\ - exp(-a * \tau) * (L - s\$1) + L\ else$

(*if* $i = 2$ *then* $\tau + s\$2$ *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*: $\bigwedge r\ ra.\ |(r::real) + - ra| = |ra + - r|$
    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*: $0 < Tmin\ Tmin \le T\ T \le Tmax$
    and *thyps*: $0 \le (\tau::real)\ \forall\tau\in\{0..\tau\}.\ \tau \le - (ln\ (Tmin\ /\ T)\ /\ a)$
  shows $Tmin \le exp\ (-a * \tau) * T$ **and** $exp\ (-a * \tau) * T \le Tmax$
proof−
  have $0 \le \tau \wedge \tau \le - (ln\ (Tmin\ /\ T)\ /\ a)$
    using *thyps* **by** *auto*
  hence $ln\ (Tmin\ /\ T) \le - a * \tau \wedge - a * \tau \le 0$
    using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
  also have $Tmin\ /\ T > 0$
    using *Thyps* **by** *auto*
  ultimately have *obs*: $Tmin\ /\ T \le exp\ (-a * \tau)\ exp\ (-a * \tau) \le 1$
    using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
  thus $Tmin \le exp\ (-a * \tau) * T$
    using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
  show $exp\ (-a * \tau) * T \le Tmax$
    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*)
⌈*I Tmin Tmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R*=λ*s. I Tmin Tmax s* ∧ *s*\$*2*=*0* ∧ *s*\$*3* = *s*\$*1* **in** *H-seq*)
apply(*rule-tac R*=λ*s. I Tmin Tmax s*∧ *s*\$*2*=*0* ∧ *s*\$*3* = *s*\$*1* **in** *H-seq*)
apply(*rule-tac R*=λ*s. I Tmin Tmax s* ∧ *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* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s. s*\$*4* = *0* ∧ *I Tmin Tmax s* ∧ *s*\$*2* = *0* ∧ *s*\$*3* = *s*\$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x*´= *f a 0* & *G Tmin Tmax a 0 on* {*0*..τ} *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* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s. s*\$*4* ≠ *0* ∧ *I Tmin Tmax s* ∧ *s*\$*2* = *0* ∧ *s*\$*3* = *s*\$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x*´= *f a L* & *G Tmin Tmax a L on* {*0*..τ} *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* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s. I Tmin Tmax s* ∧ *s*\$*2* = *0* ∧ *s*\$*3* = *s*\$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*IF* (λ*s. 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*))
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* ⌈*I Tmin Tmax*⌉ ⌈λ*s. I Tmin Tmax s* ∧ *s*\$*2* = *0*⌉ ≥ (*2* ::= (λ*s. 0*))
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-assign2*:
rel-R ⌈λ*s. I Tmin Tmax s* ∧ *s*\$*2* = *0*⌉ ⌈λ*s. I Tmin Tmax s* ∧ *s*\$*2* = *0* ∧ *s*\$*3* = *s*\$*1*⌉ ≥ (*3* ::= (λ*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* ⇒ *real^4* ⇒ *real^4* (f)
where $f\ k\ s \equiv (\chi\ i.\ \mathit{if}\ i = 2\ \mathit{then}\ 1\ \mathit{else}\ (\mathit{if}\ i = 1\ \mathit{then}\ k\ \mathit{else}\ 0))$

abbreviation *tank-flow* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (φ)
where $\varphi\ k\ \tau\ s \equiv (\chi\ i.\ \mathit{if}\ i = 1\ \mathit{then}\ k * \tau + s\$1\ \mathit{else}$
$(\mathit{if}\ i = 2\ \mathit{then}\ \tau + s\$2\ \mathit{else}\ s\$i))$

abbreviation *tank-guard* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (G)
where $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *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^4* ⇒ *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*:
$0 \leq \tau \Longrightarrow$ *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′=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 $0 \leq (\tau{::}real)$ **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 $0 \leq (\tau{::}real)$ **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 linordered-field-class.sign-simps*(*39*) *mult-less-cancel-right*)

qed

lemma *tank-inv*:
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 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*=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*)
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 $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *rel-R* ⌈*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 = λs. I hmin hmax s ∧ s$2 = 0 ∧ s$3 = s$1*
and *?cond = λs. s$4 = 0 ∧ s$3 ≤ hmin + 1*
have *ifbranch1*: *4 ::= (λs.1) ≤ rel-R ⌈λs. ?cond s ∧ ?Icntrl s⌉ ⌈?Icntrl⌉* (**is** - ≤ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) THEN (4 ::= (λs.0)) ELSE skip*) ≤
rel-R ⌈λs. ¬ ?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*) ≤ *rel-R ⌈?Icntrl⌉ ⌈?Icntrl⌉* (**is** *?ifthenelse* ≤ -)
by (*rule R-cond*)
have (*IF ?cond THEN (4 ::= (λs.1)) ELSE (IF (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) THEN (4 ::= (λs.0)) ELSE skip)*) ≤
rel-R ⌈?Icntrl⌉ ⌈?Icntrl⌉
apply(*rule-tac y=?ifthenelse* **in** *order-trans*, *rule R-cond-mono*)
using *ifbranch1 ifbranch2 ifthenelse* **by** *auto*
hence *ctrl*: *?ctrl ≤ rel-R ⌈I hmin hmax⌉ ⌈?Icntrl⌉*
apply(*rule-tac R=?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R=λs. I hmin hmax s ∧ 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))*) ≤
rel-R ⌈λs. s$4 = 0 ∧ ?Icntrl s⌉ ⌈I hmin hmax⌉
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))*) ≤
rel-R ⌈λs. s$4 ≠ 0 ∧ ?Icntrl s⌉ ⌈I hmin hmax⌉
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 ≤ rel-R ⌈?Icntrl⌉ ⌈I hmin hmax⌉*
apply(*rule order-trans*, *rule R-cond-mono*)
using *dynup dyndown* **by** (*auto intro*!: *R-cond*)
— Finally we put everything together.
have *pre-inv*: *⌈I hmin hmax⌉ ≤ ⌈I hmin hmax⌉*
by *simp*
have *inv-pos*: *⌈I hmin hmax⌉ ≤ ⌈λs. hmin ≤ s$1 ∧ s$1 ≤ hmax⌉*

by *simp*
have *inv-inv*: *rel-R* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*rel-R* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉) ≤ *rel-R* ⌈*I hmin hmax*⌉ ⌈*I hmin hmax*⌉
by (*rule R-seq*)
have *loopref*: *LOOP rel-R* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*rel-R* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉) *INV I hmin hmax* ≤ *?ref*
apply(*rule R-loop*)
using *pre-inv inv-inv inv-pos* **by** *auto*
have *obs*: *?ctrl*;*?dyn* ≤ *rel-R* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*rel-R* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉)
apply(*rule R-seq-mono*)
using *ctrl dyn* **by** *auto*
show *LOOP* (*?ctrl*;*?dyn*) *INV I hmin hmax* ≤ *?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

1.12 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

1.12.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
and *Archimedean-Field.floor-ceiling-class.floor* (⌊-⌋)
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*⌈-⌉))
where $\lceil Q \rceil \equiv (\lambda\ x{::}'a.\ \{s{::}'a.\ s = x \wedge Q\ s\})^{\bullet}$

lemma *p2ndf-simps*[*simp*]:
$\lceil P \rceil \leq \lceil Q \rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
$(\lceil P \rceil = \lceil Q \rceil) = (\forall s.\ P\ s = Q\ s)$
$(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s.\ P\ s \wedge Q\ s \rceil$
$(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s.\ P\ s \vee Q\ s \rceil$
$\mathfrak{tt}\ \lceil P \rceil = \lceil P \rceil$
$n\ \lceil P \rceil = \lceil \lambda s.\ \neg\ P\ s \rceil$
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-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*- ::= -) [*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* ⇒ *'a nd-fun* ⇒ *'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* ⇒ *'a nd-fun* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (*IF - THEN - ELSE - [64,64] 63*)
where *IF B THEN X ELSE Y* ≡ *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 $\lceil Q\rceil$ ∧ *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* ⇒ *'a pred* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (*WHILE - INV - DO - [64,64,64] 63*)
where *WHILE B INV I DO X* ≡ *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* ⇒ *'a pred* ⇒ *'a nd-fun* (*LOOP - INV - [64,64] 63*)
where *LOOP X INV I* ≡ *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*

1.12.2 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 nd-fun* (*EVOL*)
  where *EVOL* φ *G T* = (λs. *g-orbit* (λt. φ *t s*) *G T*)$^\bullet$

lemma *H-g-evol*:
  fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
  assumes *P* = (λs. ($\forall t{\in}T$. ($\forall \tau{\in}$*down T t*. *G* (φ τ *s*)) $\longrightarrow$ *Q* (φ *t s*)))
  shows *Hoare* $\lceil P \rceil$ (*EVOL* φ *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 φ :: ('*a*::*preorder*) $\Rightarrow$ '*b* $\Rightarrow$ '*b*
  shows *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* (φ τ *s*)) $\longrightarrow$ *Q* (φ *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-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 *H-g-orbital*:
  P = (λs. ($\forall X{\in}$*ivp-sols* (λt. *f*) *T S* t_0 *s*. $\forall t{\in}T$. ($\forall \tau{\in}$*down T t*. *G* (*X* τ)) $\longrightarrow$ *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-sols* (λt. *f*) *T S* t_0 *s*. $\forall t{\in}T$. ($\forall \tau{\in}$*down T t*. *G* (*X* τ)) $\longrightarrow$ *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* = (λs. *s* $\in$ *S* $\longrightarrow$ ($\forall t{\in}T$. ($\forall \tau{\in}$*down T t*. *G* (φ τ *s*)) $\longrightarrow$ *Q* (φ *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 τ] **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*: *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 $\lceil P\rceil \leq \lceil I\rceil$ **and** *Hoare* $\lceil I\rceil$ (x´= f & G on T S @ t_0) $\lceil I\rceil$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
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*)

1.12.3 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* φ *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* φ *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*: $diff\text{-}invariant \ I \ f \ T \ S \ t_0 \ G \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil \lambda s. \ I \ s \wedge G \ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$(x'{=}f \ \& \ G \ on \ T \ S \ @ \ t_0 \ DINV \ I) \leq Ref \ \lceil P \rceil \ \lceil Q \rceil$
unfolding *spec-def* **by** (*auto simp*: *H-g-ode-inv*)

1.12.4 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 $c{::}'a{::}\{heine\text{-}borel, \ banach\}$
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* $f \ T \ UNIV \ \varphi$
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 $Hoare \ \lceil P \rceil \ (x' = f \ \& \ G \ on \ T \ UNIV \ @ \ 0) \ \lceil Q \rceil$
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* $\lceil P \rceil$ ($x' = f$ & G *on* T S @ t_0) $\lceil C \rceil$
    and *wp-Q*:*Hoare* $\lceil P \rceil$ ($x' = f$ & (λ s. G s $\wedge$ C s) *on* T S @ t_0) $\lceil Q \rceil$
  shows *Hoare* $\lceil P \rceil$ ($x' = f$ & 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 ($x' = f$ & 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 nd-fun*
  ((*1x´=- & - DINV -*)) **where** ($x' = f$ & G *DINV I*) $\equiv$ ($x' = f$ & G *on UNIV UNIV* @ *0 DINV I*)

end

1.12.5 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* ⇒ *real*ˆ*2* ⇒ *real*ˆ*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*: *Hoare* ⌈λ*s*. r^2 = (*s*\$*1*)^2 + (*s*\$*2*)^2⌉ (*EVOL* φ *G T*) ⌈λ*s*. r^2 = (*s*\$*1*)^2 + (*s*\$*2*)^2⌉
by *simp*

— Verified with differential invariants

lemma *pendulum-inv*: *Hoare* ⌈λ*s*. r^2 = (*s*\$*1*)^2 + (*s*\$*2*)^2⌉ (*x*´=*f* & *G*) ⌈λ*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*: *Hoare* ⌈λ*s*. r^2 = (*s*\$*1*)^2 + (*s*\$*2*)^2⌉ (*x*´=*f* & *G*) ⌈λ*s*. r^2 = (*s*\$*1*)^2 + (*s*\$*2*)^2⌉
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 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* τ *s* ≡ (χ *i*. *if i=1 then g* · τ ˆ *2*/*2* + *s*\$*2* · τ + *s*\$*1 else g* · 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 · 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 · 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 · τ^2 / 2 + v · τ + (x::real)) =
  2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (is ?lhs = ?rhs)
proof−
  have ?lhs = g^2 · τ^2 + 2 · g · v · τ + 2 · g · x
      apply(subst Rat.sign-simps(18))+
      by(auto simp: semiring-normalization-rules(29))
    also have ... = g^2 · τ^2 + 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* $(\varphi\ g)$
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* :: $real \Rightarrow real \Rightarrow real\hat{~}4 \Rightarrow real\hat{~}4$ (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 *therm-guard* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real\hat{~}4 \Rightarrow bool$ (G)
where $G\ Tmin\ Tmax\ a\ L\ s \equiv (s\$2 \leq -\ (ln\ ((L-(if\ L=0\ then\ Tmin\ else\ Tmax))/(L-s\$3)))/a)$

abbreviation *therm-loop-inv* :: $real \Rightarrow real \Rightarrow real\hat{~}4 \Rightarrow bool$ (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* :: $real \Rightarrow real \Rightarrow real \Rightarrow real\hat{~}4 \Rightarrow real\hat{~}4$ (φ)
where $\varphi\ a\ L\ \tau\ s \equiv (\chi\ i.\ if\ i = 1\ then\ -\ exp(-a * \tau) * (L - s\$1) + L\ else\ (if\ i = 2\ then\ \tau + s\$2\ else\ s\$i))$

— 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*: $\bigwedge r\ ra.\ |(r::real) + -\ ra| = |ra + -\ r|$
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 (λt::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 ⟹ 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 ≤ T T ≤ Tmax
    and thyps: 0 ≤ (τ::real) ∀τ∈{0..τ}. τ ≤ − (ln (Tmin / T) / a)
  shows Tmin ≤ exp (−a ∗ τ) ∗ T and exp (−a ∗ τ) ∗ T ≤ Tmax
proof−
  have 0 ≤ τ ∧ τ ≤ − (ln (Tmin / T) / a)
    using thyps by auto
  hence ln (Tmin / T) ≤ − a ∗ τ ∧ − a ∗ τ ≤ 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 ∗ τ) exp (−a ∗ τ) ≤ 1
    using exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less, simp)
  thus Tmin ≤ exp (−a ∗ τ) ∗ T
    using Thyps by (simp add: pos-divide-le-eq)
  show exp (−a ∗ τ) ∗ 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 therm-dyn-up-real-arith:
  assumes a > 0 and Thyps: Tmin ≤ T T ≤ Tmax Tmax < (L::real)
    and thyps: 0 ≤ τ ∀τ∈{0..τ}. τ ≤ − (ln ((L − Tmax) / (L − T)) / a)
  shows L − Tmax ≤ exp (−(a ∗ τ)) ∗ (L − T)
    and L − exp (−(a ∗ τ)) ∗ (L − T) ≤ Tmax
    and Tmin ≤ L − exp (−(a ∗ τ)) ∗ (L − T)
proof−
  have 0 ≤ τ ∧ τ ≤ − (ln ((L − Tmax) / (L − T)) / a)
    using thyps by auto
  hence ln ((L − Tmax) / (L − T)) ≤ − a ∗ τ ∧ − a ∗ τ ≤ 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 ∗ τ) ∧ exp (−a ∗ τ) ≤ 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 *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..τ*} *UNIV @ 0*)
ELSE
(*x´= f a L & G Tmin Tmax a L on* {*0..τ*} *UNIV @ 0*))
) *INV I Tmin Tmax*)
⌈*I Tmin Tmax*⌉
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** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* ⌈$\lambda s.\ s\$4 = 0 \wedge$ *I Tmin Tmax s* $\wedge\ s\$2 = 0 \wedge s\$3 = s\$1$⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x´= f a 0 & G Tmin Tmax a 0 on* {*0..τ*} *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 ≤ τ and 0 < Tmin and Tmax < L
  shows Ref ⌈λs. s$4 ≠ 0 ∧ I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin
Tmax⌉ ≥
    (x´= f a L & G Tmin Tmax a L on {0..τ} 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 ≤ τ and 0 < Tmin and Tmax < L
  shows Ref ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉ ≥
  (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))
  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 ⌈I Tmin Tmax⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0⌉
≥ (2 ::= (λs. 0))
  by (auto simp: R-assign-rule)

lemma R-therm-assign2:
  Ref ⌈λs. I Tmin Tmax s ∧ s$2 = 0⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 =
s$1⌉ ≥ (3 ::= (λs. s$1))
  by (auto simp: R-assign-rule)

lemma R-therm-ctrl:
  Ref ⌈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*: *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* $\equiv$ (χ *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 φ *k* τ *s* $\equiv$ (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* $\Rightarrow$ *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* ⇒ *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*)

— 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* ≤ (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*)

lemmas *H-g-ode-tank* = *local-flow.sH-g-ode-ivl*[*OF local-flow-tank - UNIV-I*]

lemma *tank-flow*:
  assumes *0* ≤ τ **and** *0* < c_o **and** c_o < c_i
  shows *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*:
  $0 \leq \tau \Longrightarrow$ *diff-invariant* (*dI hmin hmax k*) (*f k*) $\{0..\tau\}$ *UNIV 0 Guard*
  apply(*intro diff-invariant-conj-rule*)
      apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
     apply(*rule-tac* $\nu'=\lambda t.\ 0$ **and** $\mu'=\lambda t.\ 1$ **in** *diff-invariant-leq-rule*, *simp-all*)
    apply(*rule-tac* $\nu'=\lambda t.\ 0$ **and** $\mu'=\lambda t.\ 0$ **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 $0 \leq (\tau::real)$ **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 $0 \leq (\tau::real)$ **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 linordered-field-class.sign-simps*(*39*) *mult-less-cancel-right*)

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* ::=($\lambda s.0$));(*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..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 $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *Ref* ⌈*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$) *on* {*0..τ*} *UNIV* @ *0 DINV* (*dI hmin hmax* ($-c_o$)))))
INV I hmin hmax) (**is** *LOOP* (*?ctrl*;*?dyn*) *INV* - ≤ *?ref*)
proof−
— First we refine the control.
let *?Icntrl* = *λs. I hmin hmax s* ∧ *s$2* = *0* ∧ *s$3* = *s$1*
and *?cond* = *λs. s$4* = *0* ∧ *s$3* ≤ *hmin* + *1*
have *ifbranch1*: *4* ::= (*λs.1*) ≤ *Ref* ⌈*λs. ?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF* (*λs. s$4* = *1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (*λs.0*)) *ELSE skip*) ≤
Ref ⌈*λs.* ¬ *?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* $\wedge$ *s\$3* $\geq$ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*)) $\leq$
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* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉
apply(*rule-tac R=?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R=*λ*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..*τ} *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$))) $\leq$
Ref ⌈λ*s. s\$4* = *0* $\wedge$ *?Icntrl s*⌉ ⌈*I hmin hmax*⌉
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 ⌈λ*s. s\$4* $\neq$ *0* $\wedge$ *?Icntrl s*⌉ ⌈*I hmin hmax*⌉
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* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉
apply(*rule order-trans*, *rule R-cond-mono*)
using *dynup dyndown* **by** (*auto intro*!: *R-cond*)
— Finally we put everything together.
have *pre-pos*: ⌈*I hmin hmax*⌉ $\leq$ ⌈*I hmin hmax*⌉
by *simp*
have *inv-inv*: *Ref* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*Ref* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉) $\leq$ *Ref* ⌈*I hmin hmax*⌉ ⌈*I hmin hmax*⌉
by (*rule R-seq*)
have *loopref*: *LOOP Ref* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*Ref* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉) *INV I hmin hmax* $\leq$ *?ref*
apply(*rule R-loop*)
using *pre-pos inv-inv* **by** *auto*
have *obs*: *?ctrl*;*?dyn* $\leq$ *Ref* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*Ref* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉)
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

1.13 Kleene Algebras

theory *Kleene-Algebra*
imports *Kleene-Algebra.Conway*
begin

1.13.1 Left Near Kleene Algebras

Extending the hierarchy developed in *Kleene-Algebra.Dioid* we now add an operation of Kleene star, finite iteration, or reflexive transitive closure to variants of Dioids. Since a multiplicative unit is needed for defining the star we only consider variants with 1; 0 can be added separately. We consider the left star induction axiom and the right star induction axiom independently since in some applications, e.g., Salomaa's axioms, probabilistic Kleene algebras, or completeness proofs with respect to the equational theoy of regular expressions and regular languages, the right star induction axiom is not needed or not valid.

We start with near dioids, then consider pre-dioids and finally dioids. It turns out that many of the known laws of Kleene algebras hold already in these more general settings. In fact, all our equational theorems have been proved within left Kleene algebras, as expected.

Although most of the proofs in this file could be fully automated by Sledgehammer and Metis, we display step-wise proofs as they would appear in a text book. First, this file may then be useful as a reference manual on Kleene algebra. Second, it is better protected against changes in the underlying theories and supports easy translation of proofs into other settings.

class *left-near-kleene-algebra* = *near-dioid-one* + *star-op* +
 assumes *star-unfoldl*: $1 + x \cdot x^\star \leq x^\star$
 and *star-inductl*: $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$

begin

First we prove two immediate consequences of the unfold axiom. The first one states that starred elements are reflexive.

lemma *star-ref* [*simp*]: $1 \leq x^\star$
 using *star-unfoldl* **by** *auto*

Reflexivity of starred elements implies, by definition of the order, that 1 is an additive unit for starred elements.

lemma *star-plus-one* [*simp*]: $1 + x^\star = x^\star$
 using *less-eq-def star-ref* **by** *blast*

lemma *star-1l* [*simp*]: $x \cdot x^\star \leq x^\star$
 using *star-unfoldl* **by** *auto*

lemma $x^\star \cdot x \leq x^\star$

oops

lemma $x \cdot x^\star = x^\star$

oops

Next we show that starred elements are transitive.

lemma *star-trans-eq* [*simp*]: $x^\star \cdot x^\star = x^\star$
proof (*rule antisym*) — this splits an equation into two inequalities
  have $x^\star + x \cdot x^\star \leq x^\star$
    by *auto*
  thus $x^\star \cdot x^\star \leq x^\star$
    by (*simp add*: *star-inductl*)
  next show $x^\star \leq x^\star \cdot x^\star$
    using *mult-isor star-ref* **by** *fastforce*
qed

lemma *star-trans*: $x^\star \cdot x^\star \leq x^\star$
  by *simp*

We now derive variants of the star induction axiom.

lemma *star-inductl-var*: $x \cdot y \leq y \Longrightarrow x^\star \cdot y \leq y$
proof −
  assume $x \cdot y \leq y$
  hence $y + x \cdot y \leq y$
    by *simp*
  thus $x^\star \cdot y \leq y$
    by (*simp add*: *star-inductl*)
qed

lemma *star-inductl-var-equiv* [*simp*]: $x^\star \cdot y \leq y \longleftrightarrow x \cdot y \leq y$
proof
  assume $x \cdot y \leq y$
  thus $x^\star \cdot y \leq y$
    by (*simp add*: *star-inductl-var*)
next
  assume $x^\star \cdot y \leq y$
  hence $x^\star \cdot y = y$
    by (*metis eq-iff mult-1-left mult-isor star-ref*)
  moreover hence $x \cdot y = x \cdot x^\star \cdot y$
    by (*simp add*: *mult.assoc*)
  moreover have ... $\leq x^\star \cdot y$
    by (*metis mult-isor star-1l*)
  ultimately show $x \cdot y \leq y$
    by *auto*
qed

lemma *star-inductl-var-eq*: $x \cdot y = y \Longrightarrow x^\star \cdot y \leq y$

by (*metis eq-iff star-inductl-var*)

lemma *star-inductl-var-eq2*: $y = x \cdot y \Longrightarrow y = x^\star \cdot y$
proof –
  assume *hyp*: $y = x \cdot y$
  hence $y \leq x^\star \cdot y$
    using *mult-isor star-ref* **by** *fastforce*
  thus $y = x^\star \cdot y$
    using *hyp eq-iff* **by** *auto*
qed

lemma $y = x \cdot y \longleftrightarrow y = x^\star \cdot y$

  oops

lemma $x^\star \cdot z \leq y \Longrightarrow z + x \cdot y \leq y$

  oops

lemma *star-inductl-one*: $1 + x \cdot y \leq y \Longrightarrow x^\star \leq y$
  using *star-inductl* **by** *force*

lemma *star-inductl-star*: $x \cdot y^\star \leq y^\star \Longrightarrow x^\star \leq y^\star$
  by (*simp add*: *star-inductl-one*)

lemma *star-inductl-eq*: $z + x \cdot y = y \Longrightarrow x^\star \cdot z \leq y$
  by (*simp add*: *star-inductl*)

We now prove two facts related to 1.

lemma *star-subid*: $x \leq 1 \Longrightarrow x^\star = 1$
proof –
  assume $x \leq 1$
  hence $1 + x \cdot 1 \leq 1$
    by *simp*
  hence $x^\star \leq 1$
    by (*metis mult-oner star-inductl*)
  thus $x^\star = 1$
    by (*simp add*: *order.antisym*)
qed

lemma *star-one* [*simp*]: $1^\star = 1$
  by (*simp add*: *star-subid*)

We now prove a subdistributivity property for the star (which is equivalent to isotonicity of star).

lemma *star-subdist*: $x^\star \leq (x + y)^\star$
proof –
  have $x \cdot (x + y)^\star \leq (x + y) \cdot (x + y)^\star$
    by *simp*

also have ... $\leq (x + y)^\star$
by (*metis star-1l*)
thus *?thesis*
using *calculation order-trans star-inductl-star* **by** *blast*
qed

lemma *star-subdist-var*: $x^\star + y^\star \leq (x + y)^\star$
using *join.sup-commute star-subdist* **by** *force*

lemma *star-iso* [*intro*]: $x \leq y \Longrightarrow x^\star \leq y^\star$
by (*metis less-eq-def star-subdist*)

We now prove some more simple properties.

lemma *star-invol* [*simp*]: $(x^\star)^\star = x^\star$
proof (*rule antisym*)
have $x^\star \cdot x^\star = x^\star$
by (*fact star-trans-eq*)
thus $(x^\star)^\star \leq x^\star$
by (*simp add*: *star-inductl-star*)
have$(x^\star)^\star \cdot (x^\star)^\star \leq (x^\star)^\star$
by (*fact star-trans*)
hence $x \cdot (x^\star)^\star \leq (x^\star)^\star$
by (*meson star-inductl-var-equiv*)
thus $x^\star \leq (x^\star)^\star$
by (*simp add*: *star-inductl-star*)
qed

lemma *star2* [*simp*]: $(1 + x)^\star = x^\star$
proof (*rule antisym*)
show $x^\star \leq (1 + x)^\star$
by *auto*
have $x^\star + x \cdot x^\star \leq x^\star$
by *simp*
thus $(1 + x)^\star \leq x^\star$
by (*simp add*: *star-inductl-star*)
qed

lemma $1 + x^\star \cdot x \leq x^\star$

oops

lemma $x \leq x^\star$

oops

lemma $x^\star \cdot x \leq x^\star$

oops

lemma $1 + x \cdot x^\star = x^\star$

oops

lemma $x \cdot z \leq z \cdot y \Longrightarrow x^\star \cdot z \leq z \cdot y^\star$

oops

The following facts express inductive conditions that are used to show that $(x + y)^\star$ is the greatest term that can be built from x and y.

lemma *prod-star-closure*: $x \leq z^\star \Longrightarrow y \leq z^\star \Longrightarrow x \cdot y \leq z^\star$
proof −
assume *assm*: $x \leq z^\star$ $y \leq z^\star$
hence $y + z^\star \cdot z^\star \leq z^\star$
by *simp*
hence $z^\star \cdot y \leq z^\star$
by (*simp add*: *star-inductl*)
also have $x \cdot y \leq z^\star \cdot y$
by (*simp add*: *assm mult-isor*)
thus $x \cdot y \leq z^\star$
using *calculation order.trans* **by** *blast*
qed

lemma *star-star-closure*: $x^\star \leq z^\star \Longrightarrow (x^\star)^\star \leq z^\star$
by (*metis star-invol*)

lemma *star-closed-unfold*: $x^\star = x \Longrightarrow x = 1 + x \cdot x$
by (*metis star-plus-one star-trans-eq*)

lemma $x^\star = x \longleftrightarrow x = 1 + x \cdot x$

oops

end

1.13.2 Left Pre-Kleene Algebras

class *left-pre-kleene-algebra* = *left-near-kleene-algebra* + *pre-dioid-one*

begin

We first prove that the star operation is extensive.

lemma *star-ext* [*simp*]: $x \leq x^\star$
proof −
have $x \leq x \cdot x^\star$
by (*metis mult-oner mult-isol star-ref*)
thus *?thesis*
by (*metis order-trans star-1l*)
qed

We now prove a right star unfold law.

lemma *star-1r* [*simp*]: $x^\star \cdot x \leq x^\star$
proof $-$
  have $x + x \cdot x^\star \leq x^\star$
    by *simp*
  thus *?thesis*
    by (*fact star-inductl*)
qed

lemma *star-unfoldr*: $1 + x^\star \cdot x \leq x^\star$
  by *simp*

lemma $1 + x^\star \cdot x = x^\star$

  oops

Next we prove a simulation law for the star. It is instrumental in proving further properties.

lemma *star-sim1*: $x \cdot z \leq z \cdot y \Longrightarrow x^\star \cdot z \leq z \cdot y^\star$
proof $-$
  assume $x \cdot z \leq z \cdot y$
  hence $x \cdot z \cdot y^\star \leq z \cdot y \cdot y^\star$
    by (*simp add*: *mult-isor*)
  also have ... $\leq z \cdot y^\star$
    by (*simp add*: *mult-isol mult-assoc*)
  finally have $x \cdot z \cdot y^\star \leq z \cdot y^\star$
    by *simp*
  hence $z + x \cdot z \cdot y^\star \leq z \cdot y^\star$
    by (*metis join.sup-least mult-isol mult-oner star-ref*)
  thus $x^\star \cdot z \leq z \cdot y^\star$
    by (*simp add*: *star-inductl mult-assoc*)
qed

The next lemma is used in omega algebras to prove, for instance, Bachmair and Dershowitz's separation of termination theorem [**?**]. The property at the left-hand side of the equivalence is known as *quasicommutation*.

lemma *quasicomm-var*: $y \cdot x \leq x \cdot (x + y)^\star \longleftrightarrow y^\star \cdot x \leq x \cdot (x + y)^\star$
proof
  assume $y \cdot x \leq x \cdot (x + y)^\star$
  thus $y^\star \cdot x \leq x \cdot (x + y)^\star$
    using *star-sim1* **by** *force*
next
  assume $y^\star \cdot x \leq x \cdot (x + y)^\star$
  thus $y \cdot x \leq x \cdot (x + y)^\star$
    by (*meson mult-isor order-trans star-ext*)
qed

lemma *star-slide1*: $(x \cdot y)^\star \cdot x \leq x \cdot (y \cdot x)^\star$

by (*simp add*: *mult-assoc star-sim1*)

lemma $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$

oops

lemma *star-slide-var1*: $x^\star \cdot x \leq x \cdot x^\star$
by (*simp add*: *star-sim1*)

We now show that the (left) star unfold axiom can be strengthened to an equality.

lemma *star-unfoldl-eq* [*simp*]: $1 + x \cdot x^\star = x^\star$
proof (*rule antisym*)
show $1 + x \cdot x^\star \leq x^\star$
by (*fact star-unfoldl*)
have $1 + x \cdot (1 + x \cdot x^\star) \leq 1 + x \cdot x^\star$
by (*meson join.sup-mono eq-refl mult-isol star-unfoldl*)
thus $x^\star \leq 1 + x \cdot x^\star$
by (*simp add*: *star-inductl-one*)
qed

lemma $1 + x^\star \cdot x = x^\star$

oops

Next we relate the star and the reflexive transitive closure operation.

lemma *star-rtc1-eq* [*simp*]: $1 + x + x^\star \cdot x^\star = x^\star$
by (*simp add*: *join.sup.absorb2*)

lemma *star-rtc1*: $1 + x + x^\star \cdot x^\star \leq x^\star$
by *simp*

lemma *star-rtc2*: $1 + x \cdot x \leq x \longleftrightarrow x = x^\star$
proof
assume $1 + x \cdot x \leq x$
thus $x = x^\star$
by (*simp add*: *local.eq-iff local.star-inductl-one*)
next
assume $x = x^\star$
thus $1 + x \cdot x \leq x$
using *local.star-closed-unfold* **by** *auto*
qed

lemma *star-rtc3*: $1 + x \cdot x = x \longleftrightarrow x = x^\star$
by (*metis order-refl star-plus-one star-rtc2 star-trans-eq*)

lemma *star-rtc-least*: $1 + x + y \cdot y \leq y \Longrightarrow x^\star \leq y$
proof –
assume $1 + x + y \cdot y \leq y$

hence *1* $+ x \cdot y \leq y$
by (*metis join.le-sup-iff mult-isol-var star-trans-eq star-rtc2*)
thus $x^\star \leq y$
by (*simp add*: *star-inductl-one*)
qed

lemma *star-rtc-least-eq*: *1* $+ x + y \cdot y = y \Longrightarrow x^\star \leq y$
by (*simp add*: *star-rtc-least*)

lemma *1* $+ x + y \cdot y \leq y \longleftrightarrow x^\star \leq y$

oops

The next lemmas are again related to closure conditions

lemma *star-subdist-var-1*: $x \leq (x + y)^\star$
by (*meson join.sup.boundedE star-ext*)

lemma *star-subdist-var-2*: $x \cdot y \leq (x + y)^\star$
by (*meson join.sup.boundedE prod-star-closure star-ext*)

lemma *star-subdist-var-3*: $x^\star \cdot y^\star \leq (x + y)^\star$
by (*simp add*: *prod-star-closure star-iso*)

We now prove variants of sum-elimination laws under a star. These are also known a denesting laws or as sum-star laws.

lemma *star-denest* [*simp*]: $(x + y)^\star = (x^\star \cdot y^\star)^\star$
proof (*rule antisym*)
have $x + y \leq x^\star \cdot y^\star$
by (*metis join.sup.bounded-iff mult-1-right mult-isol-var mult-onel star-ref star-ext*)
thus $(x + y)^\star \leq (x^\star \cdot y^\star)^\star$
by (*fact star-iso*)
have $x^\star \cdot y^\star \leq (x + y)^\star$
by (*fact star-subdist-var-3*)
thus $(x^\star \cdot y^\star)^\star \leq (x + y)^\star$
by (*simp add*: *prod-star-closure star-inductl-star*)
qed

lemma *star-sum-var* [*simp*]: $(x^\star + y^\star)^\star = (x + y)^\star$
by *simp*

lemma *star-denest-var* [*simp*]: $x^\star \cdot (y \cdot x^\star)^\star = (x + y)^\star$
proof (*rule antisym*)
have $1 \leq x^\star \cdot (y \cdot x^\star)^\star$
by (*metis mult-isol-var mult-oner star-ref*)
moreover have $x \cdot x^\star \cdot (y \cdot x^\star)^\star \leq x^\star \cdot (y \cdot x^\star)^\star$
by (*simp add*: *mult-isor*)
moreover have $y \cdot x^\star \cdot (y \cdot x^\star)^\star \leq x^\star \cdot (y \cdot x^\star)^\star$
by (*metis mult-isol-var mult-onel star-1l star-ref*)

ultimately have $1 + (x + y) \cdot x^\star \cdot (y \cdot x^\star)^\star \leq x^\star \cdot (y \cdot x^\star)^\star$
by *auto*
thus $(x + y)^\star \leq x^\star \cdot (y \cdot x^\star)^\star$
by (*metis mult.assoc mult-oner star-inductl*)
have $(y \cdot x^\star)^\star \leq (y^\star \cdot x^\star)^\star$
by (*simp add*: *mult-isol-var star-iso*)
hence $(y \cdot x^\star)^\star \leq (x + y)^\star$
by (*metis add.commute star-denest*)
moreover have $x^\star \leq (x + y)^\star$
by (*fact star-subdist*)
ultimately show $x^\star \cdot (y \cdot x^\star)^\star \leq (x + y)^\star$
using *prod-star-closure* **by** *blast*
qed

lemma *star-denest-var-2* [*simp*]: $x^\star \cdot (y \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
by *simp*

lemma *star-denest-var-3* [*simp*]: $x^\star \cdot (y^\star \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
by *simp*

lemma *star-denest-var-4* [*ac-simps*]: $(y^\star \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
by (*metis add-comm star-denest*)

lemma *star-denest-var-5* [*ac-simps*]: $x^\star \cdot (y \cdot x^\star)^\star = y^\star \cdot (x \cdot y^\star)^\star$
by (*simp add*: *star-denest-var-4*)

lemma $x^\star \cdot (y \cdot x^\star)^\star = (x^\star \cdot y)^\star \cdot x^\star$

oops

lemma *star-denest-var-6* [*simp*]: $x^\star \cdot y^\star \cdot (x + y)^\star = (x + y)^\star$
using *mult-assoc* **by** *simp*

lemma *star-denest-var-7* [*simp*]: $(x + y)^\star \cdot x^\star \cdot y^\star = (x + y)^\star$
proof (*rule antisym*)
have $(x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot (x^\star \cdot y^\star)^\star$
by (*simp add*: *mult-assoc*)
thus $(x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star$
by *simp*
have $1 \leq (x + y)^\star \cdot x^\star \cdot y^\star$
by (*metis dual-order.trans mult-1-left mult-isor star-ref*)
moreover have $(x + y) \cdot (x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
using *mult-isor star-1l* **by** *presburger*
ultimately have $1 + (x + y) \cdot (x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
by *simp*
thus $(x + y)^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
by (*metis mult.assoc star-inductl-one*)
qed

lemma *star-denest-var-8* [*simp*]: $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star)^\star = (x^\star \cdot y^\star)^\star$
  by (*simp add*: *mult-assoc*)

lemma *star-denest-var-9* [*simp*]: $(x^\star \cdot y^\star)^\star \cdot x^\star \cdot y^\star = (x^\star \cdot y^\star)^\star$
  using *star-denest-var-7* **by** *simp*

The following statements are well known from term rewriting. They are all variants of the Church-Rosser theorem in Kleene algebra [**?**]. But first we prove a law relating two confluence properties.

lemma *confluence-var* [*iff*]: $y \cdot x^\star \leq x^\star \cdot y^\star \longleftrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$
proof
  assume $y \cdot x^\star \leq x^\star \cdot y^\star$
  thus $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
    using *star-sim1* **by** *fastforce*
next
  assume $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
  thus $y \cdot x^\star \leq x^\star \cdot y^\star$
    by (*meson mult-isor order-trans star-ext*)
qed

lemma *church-rosser* [*intro*]: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
proof (*rule antisym*)
  assume $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
  hence $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot x^\star \cdot y^\star \cdot y^\star$
    by (*metis mult-isol mult-isor mult.assoc*)
  hence $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot y^\star$
    by (*simp add*: *mult-assoc*)
  moreover have $1 \leq x^\star \cdot y^\star$
    by (*metis dual-order.trans mult-1-right mult-isol star-ref*)
  ultimately have $1 + x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot y^\star$
    by *simp*
  hence $(x^\star \cdot y^\star)^\star \leq x^\star \cdot y^\star$
    by (*simp add*: *star-inductl-one*)
  thus $(x + y)^\star \leq x^\star \cdot y^\star$
    by *simp*
  thus $x^\star \cdot y^\star \leq (x + y)^\star$
    by *simp*
qed

lemma *church-rosser-var*: $y \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
  by *fastforce*

lemma *church-rosser-to-confluence*: $(x + y)^\star \leq x^\star \cdot y^\star \Longrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$
  by (*metis add-comm eq-iff star-subdist-var-3*)

lemma *church-rosser-equiv*: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \longleftrightarrow (x + y)^\star = x^\star \cdot y^\star$
  using *church-rosser-to-confluence eq-iff* **by** *blast*

lemma *confluence-to-local-confluence*: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow y \cdot x \leq x^\star \cdot y^\star$
  by (*meson mult-isol-var order-trans star-ext*)

lemma $y \cdot x \leq x^\star \cdot y^\star \Longrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$

  oops

lemma $y \cdot x \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star \leq x^\star \cdot y^\star$
  oops

More variations could easily be proved. The last counterexample shows that Newman's lemma needs a wellfoundedness assumption. This is well known.

The next lemmas relate the reflexive transitive closure and the transitive closure.

lemma *sup-id-star1*: $1 \leq x \Longrightarrow x \cdot x^\star = x^\star$
proof −
  assume $1 \leq x$
  hence $x^\star \leq x \cdot x^\star$
    using *mult-isor* **by** *fastforce*
  thus $x \cdot x^\star = x^\star$
    by (*simp add*: *eq-iff*)
qed

lemma *sup-id-star2*: $1 \leq x \Longrightarrow x^\star \cdot x = x^\star$
  by (*metis order.antisym mult-isol mult-oner star-1r*)

lemma $1 + x^\star \cdot x = x^\star$

  oops

lemma $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$

  oops

lemma $x \cdot x = x \Longrightarrow x^\star = 1 + x$

  oops

end

1.13.3 Left Kleene Algebras

class *left-kleene-algebra* = *left-pre-kleene-algebra* + *dioid-one*

begin

In left Kleene algebras the non-fact $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$ is a good challenge for counterexample generators. A model of left Kleene algebras in which the right star induction law does not hold has been given by Kozen [**?**].

We now show that the right unfold law becomes an equality.

lemma *star-unfoldr-eq* [*simp*]: $1 + x^\star \cdot x = x^\star$
proof (*rule antisym*)
  show $1 + x^\star \cdot x \leq x^\star$
    by (*fact star-unfoldr*)
  have $1 + x \cdot (1 + x^\star \cdot x) = 1 + (1 + x \cdot x^\star) \cdot x$
    using *distrib-left distrib-right mult-1-left mult-1-right mult-assoc* **by** *presburger*
  also have ... $= 1 + x^\star \cdot x$
    by *simp*
  finally show $x^\star \leq 1 + x^\star \cdot x$
    by (*simp add*: *star-inductl-one*)
qed

The following more complex unfold law has been used as an axiom, called prodstar, by Conway [**?**].

lemma *star-prod-unfold* [*simp*]: $1 + x \cdot (y \cdot x)^\star \cdot y = (x \cdot y)^\star$
proof (*rule antisym*)
  have $(x \cdot y)^\star = 1 + (x \cdot y)^\star \cdot x \cdot y$
    by (*simp add*: *mult-assoc*)
  thus $(x \cdot y)^\star \leq 1 + x \cdot (y \cdot x)^\star \cdot y$
    by (*metis join.sup-mono mult-isor order-refl star-slide1*)
  have $1 + x \cdot (y \cdot x)^\star \cdot y \leq 1 + x \cdot y \cdot (x \cdot y)^\star$
    by (*metis join.sup-mono eq-refl mult.assoc mult-isol star-slide1*)
  thus $1 + x \cdot (y \cdot x)^\star \cdot y \leq (x \cdot y)^\star$
    by *simp*
qed

The slide laws, which have previously been inequalities, now become equations.

lemma *star-slide* [*ac-simps*]: $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$
proof −
  have $x \cdot (y \cdot x)^\star = x \cdot (1 + y \cdot (x \cdot y)^\star \cdot x)$
    by *simp*
  also have ... $= (1 + x \cdot y \cdot (x \cdot y)^\star) \cdot x$
    by (*simp add*: *distrib-left mult-assoc*)
  finally show *?thesis*
    by *simp*
qed

lemma *star-slide-var* [*ac-simps*]: $x^\star \cdot x = x \cdot x^\star$
  by (*metis mult-onel mult-oner star-slide*)

lemma *star-sum-unfold-var* [*simp*]: $1 + x^\star \cdot (x + y)^\star \cdot y^\star = (x + y)^\star$
  by (*metis star-denest star-denest-var-3 star-denest-var-4 star-plus-one star-slide*)

The following law shows how starred sums can be unfolded.

lemma *star-sum-unfold* [*simp*]: $x^\star + x^\star \cdot y \cdot (x + y)^\star = (x + y)^\star$
proof −
  have $(x + y)^\star = x^\star \cdot (y \cdot x^\star\ )^\star$
    by *simp*
  also have ... $= x^\star \cdot (1 + y \cdot x^\star \cdot (y \cdot x^\star\ )^\star)$
    by *simp*
  also have ... $= x^\star \cdot (1 + y \cdot (x + y)^\star)$
    by (*simp add*: *mult.assoc*)
  finally show *?thesis*
    by (*simp add*: *distrib-left mult-assoc*)
qed

The following property appears in process algebra.

lemma *troeger*: $(x + y)^\star \cdot z = x^\star \cdot (y \cdot (x + y)^\star \cdot z + z)$
proof −
  have $(x + y)^\star \cdot z = x^\star \cdot z + x^\star \cdot y \cdot (x + y)^\star \cdot z$
    by (*metis* (*full-types*) *distrib-right star-sum-unfold*)
  thus *?thesis*
    by (*simp add*: *add-commute distrib-left mult-assoc*)
qed

The following properties are related to a property from propositional dynamic logic which has been attributed to Albert Meyer [**?**]. Here we prove it as a theorem of Kleene algebra.

lemma *star-square*: $(x \cdot x)^\star \leq x^\star$
proof −
  have $x \cdot x \cdot x^\star \leq x^\star$
    by (*simp add*: *prod-star-closure*)
  thus *?thesis*
    by (*simp add*: *star-inductl-star*)
qed

lemma *meyer-1* [*simp*]: $(1 + x) \cdot (x \cdot x)^\star = x^\star$
proof (*rule antisym*)
  have $x \cdot\ (1 + x) \cdot (x \cdot x)^\star = x \cdot (x \cdot x)^\star + x \cdot x \cdot (x \cdot x)^\star$
    by (*simp add*: *distrib-left*)
  also have ... $\leq x \cdot (x \cdot x)^\star + (x \cdot x)^\star$
    using *join.sup-mono star-1l* **by** *blast*
  finally have $x \cdot\ (1 + x) \cdot (x \cdot x)^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by (*simp add*: *join.sup-commute*)
  moreover have $1 \leq (1 + x) \cdot (x \cdot x)^\star$
    using *join.sup.coboundedI1* **by** *auto*
  ultimately have $1 + x \cdot (1 + x) \cdot (x \cdot x)^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by *auto*
  thus $x^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by (*simp add*: *star-inductl-one mult-assoc*)
  show $(1 + x) \cdot (x \cdot x)^\star \leq x^\star$
    by (*simp add*: *prod-star-closure star-square*)
qed

The following lemma says that transitive elements are equal to their transitive closure.

lemma *tc*: $x \cdot x \leq x \Longrightarrow x^\star \cdot x = x$
proof –
  assume $x \cdot x \leq x$
  hence $x + x \cdot x \leq x$
    by *simp*
  hence $x^\star \cdot x \leq x$
    by (*fact star-inductl*)
  thus $x^\star \cdot x = x$
    by (*metis mult-isol mult-oner star-ref star-slide-var eq-iff*)
qed

lemma *tc-eq*: $x \cdot x = x \Longrightarrow x^\star \cdot x = x$
  by (*auto intro*: *tc*)

The next fact has been used by Boffa [**?**] to axiomatise the equational theory of regular expressions.

lemma *boffa-var*: $x \cdot x \leq x \Longrightarrow x^\star = 1 + x$
proof –
  assume $x \cdot x \leq x$
  moreover have $x^\star = 1 + x^\star \cdot x$
    by *simp*
  ultimately show $x^\star = 1 + x$
    by (*simp add*: *tc*)
qed

lemma *boffa*: $x \cdot x = x \Longrightarrow x^\star = 1 + x$
  by (*auto intro*: *boffa-var*)

end

1.13.4 Left Kleene Algebras with Zero

There are applications where only a left zero is assumed, for instance in the context of total correctness and for demonic refinement algebras [**?**].

class *left-kleene-algebra-zerol* = *left-kleene-algebra* + *dioid-one-zerol*
begin

sublocale *conway*: *near-conway-base-zerol star*
  by *standard* (*simp-all add*: *local.star-slide*)

lemma *star-zero* [*simp*]: $0^\star = 1$
  by (*rule local.conway.zero-dagger*)

In principle, 1 could therefore be defined from 0 in this setting.

end

class *left-kleene-algebra-zero* = *left-kleene-algebra-zerol* + *dioid-one-zero*

1.13.5 Pre-Kleene Algebras

Pre-Kleene algebras are essentially probabilistic Kleene algebras [**?**]. They have a weaker right star unfold axiom. We are still looking for theorems that could be proved in this setting.

class *pre-kleene-algebra* = *left-pre-kleene-algebra* +
  assumes *weak-star-unfoldr*: $z + y \cdot (x + 1) \leq y \Longrightarrow z \cdot x^\star \leq y$

1.13.6 Kleene Algebras

class *kleene-algebra-zerol* = *left-kleene-algebra-zerol* +
  assumes *star-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$

begin

lemma *star-sim2*: $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\star \leq y^\star \cdot z$
proof –
  assume $z \cdot x \leq y \cdot z$
  hence $y^\star \cdot z \cdot x \leq y^\star \cdot y \cdot z$
    using *mult-isol mult-assoc* **by** *auto*
  also have ... $\leq y^\star \cdot z$
    by (*simp add*: *mult-isor*)
  finally have $y^\star \cdot z \cdot x \leq y^\star \cdot z$
    by *simp*
  moreover have $z \leq y^\star \cdot z$
    using *mult-isor star-ref* **by** *fastforce*
  ultimately have $z + y^\star \cdot z \cdot x \leq y^\star \cdot z$
    by *simp*
  thus $z \cdot x^\star \leq y^\star \cdot z$
    by (*simp add*: *star-inductr*)
qed

sublocale *conway*: *pre-conway star*
  by *standard* (*simp add*: *star-sim2*)

lemma *star-inductr-var*: $y \cdot x \leq y \Longrightarrow y \cdot x^\star \leq y$
  by (*simp add*: *star-inductr*)

lemma *star-inductr-var-equiv*: $y \cdot x \leq y \longleftrightarrow y \cdot x^\star \leq y$
  by (*meson order-trans mult-isol star-ext star-inductr-var*)

lemma *star-sim3*: $z \cdot x = y \cdot z \Longrightarrow z \cdot x^\star = y^\star \cdot z$
  by (*simp add*: *eq-iff star-sim1 star-sim2*)

lemma *star-sim4*: $x \cdot y \leq\ y \cdot x \Longrightarrow x^\star \cdot y^\star \leq y^\star \cdot x^\star$
  by (*auto intro*: *star-sim1 star-sim2*)

lemma *star-inductr-eq*: $z + y \cdot x = y \Longrightarrow z \cdot x^\star \leq y$
  by (*auto intro*: *star-inductr*)

lemma *star-inductr-var-eq*: $y \cdot x = y \Longrightarrow y \cdot x^\star \leq y$
  by (*auto intro*: *star-inductr-var*)

lemma *star-inductr-var-eq2*: $y \cdot x = y \Longrightarrow y \cdot x^\star = y$
  by (*metis mult-onel star-one star-sim3*)

lemma *bubble-sort*: $y \cdot x \leq x \cdot y \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
  by (*fastforce intro*: *star-sim4*)

lemma *independence1*: $x \cdot y = 0 \Longrightarrow x^\star \cdot y = y$
proof −
  assume $x \cdot y = 0$
  moreover have $x^\star \cdot y = y + x^\star \cdot x \cdot y$
    by (*metis distrib-right mult-onel star-unfoldr-eq*)
  ultimately show $x^\star \cdot y = y$
    by (*metis add-0-left add.commute join.sup-ge1 eq-iff star-inductl-eq*)
qed

lemma *independence2*: $x \cdot y = 0 \Longrightarrow x \cdot y^\star = x$
  by (*metis annil mult-onel star-sim3 star-zero*)

lemma *lazycomm-var*: $y \cdot x \leq x \cdot (x + y)^\star + y \longleftrightarrow y \cdot x^\star \leq x \cdot (x + y)^\star + y$
proof
  let $?t = x \cdot (x + y)^\star$
  assume *hyp*: $y \cdot x \leq ?t + y$
  have $(?t + y) \cdot x = ?t \cdot x + y \cdot x$
    by (*fact distrib-right*)
  also have ... $\leq ?t \cdot x + ?t + y$
    using *hyp join.sup.coboundedI2 join.sup-assoc* **by** *auto*
  also have ... $\leq ?t + y$
   using *eq-refl join.sup-least join.sup-mono mult-isol prod-star-closure star-subdist-var-1 mult-assoc* **by** *presburger*
  finally have $y + (?t + y) \cdot x \leq ?t + y$
    by *simp*
  thus $y \cdot x^\star \leq x \cdot (x + y)^\star + y$
    by (*fact star-inductr*)
next
  assume $y \cdot x^\star \leq x \cdot (x + y)^\star + y$
  thus $y \cdot x \leq x \cdot (x + y)^\star + y$
    using *dual-order.trans mult-isol star-ext* **by** *blast*
qed

lemma *arden-var*: $(\forall y\ v.\ y \leq x \cdot y + v \longrightarrow y \leq x^\star \cdot v) \Longrightarrow z = x \cdot z + w \Longrightarrow$

$z = x^\star \cdot w$
  by (*auto simp*: *add-comm eq-iff star-inductl-eq*)

lemma $(\forall x\ y.\ y \leq x \cdot y \longrightarrow y = 0) \Longrightarrow y \leq x \cdot y + z \Longrightarrow y \leq x^\star \cdot z$
  by (*metis eq-refl mult-onel*)

end

Finally, here come the Kleene algebras à la Kozen [**?**]. We only prove quasi-identities in this section. Since left Kleene algebras are complete with respect to the equational theory of regular expressions and regular languages, all identities hold already without the right star induction axiom.

class *kleene-algebra* = *left-kleene-algebra-zero* +
  assumes *star-inductr'*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
begin

subclass *kleene-algebra-zerol*
  by *standard* (*simp add*: *star-inductr'*)

sublocale *conway-zerol*: *conway star* **..**

The next lemma shows that opposites of Kleene algebras (i.e., Kleene algebras with the order of multiplication swapped) are again Kleene algebras.

lemma *dual-kleene-algebra*:
  class.kleene-algebra $(+)$ $(\odot)$ 1 0 $(\leq)$ $(<)$ *star*
proof
  fix $x\ y\ z :: 'a$
  show $(x \odot y) \odot z = x \odot (y \odot z)$
    by (*metis mult.assoc opp-mult-def*)
  show $(x + y) \odot z = x \odot z + y \odot z$
    by (*metis opp-mult-def distrib-left*)
  show $1 \odot x = x$
    by (*metis mult-oner opp-mult-def*)
  show $x \odot 1 = x$
    by (*metis mult-onel opp-mult-def*)
  show $0 + x = x$
    by (*fact add-zerol*)
  show $0 \odot x = 0$
    by (*metis annir opp-mult-def*)
  show $x \odot 0 = 0$
    by (*metis annil opp-mult-def*)
  show $x + x = x$
    by (*fact add-idem*)
  show $x \odot (y + z) = x \odot y + x \odot z$
    by (*metis distrib-right opp-mult-def*)
  show $z \odot x \leq z \odot (x + y)$
    by (*metis mult-isor opp-mult-def order-prop*)
  show $1 + x \odot x^\star \leq x^\star$
    by (*metis opp-mult-def order-refl star-slide-var star-unfoldl-eq*)

show $z + x \odot y \leq y \Longrightarrow x^\star \odot z \leq y$
by (*metis opp-mult-def star-inductr*)
show $z + y \odot x \leq y \Longrightarrow z \odot x^\star \leq y$
by (*metis opp-mult-def star-inductl*)
qed

end

We finish with some properties on (multiplicatively) commutative Kleene algebras. A chapter in Conway's book [**?**] is devoted to this topic.

class *commutative-kleene-algebra* = *kleene-algebra* +
assumes *mult-comm* [*ac-simps*]: $x \cdot y = y \cdot x$

begin

lemma *conway-c3* [*simp*]: $(x + y)^\star = x^\star \cdot y^\star$
using *church-rosser mult-comm* **by** *auto*

lemma *conway-c4*: $(x^\star \cdot y)^\star = 1 + x^\star \cdot y^\star \cdot y$
by (*metis conway-c3 star-denest-var star-prod-unfold*)

lemma *cka-1*: $(x \cdot y)^\star \leq x^\star \cdot y^\star$
by (*metis conway-c3 star-invol star-iso star-subdist-var-2*)

lemma *cka-2* [*simp*]: $x^\star \cdot (x^\star \cdot y)^\star = x^\star \cdot y^\star$
by (*metis conway-c3 mult-comm star-denest-var*)

lemma *conway-c4-var* [*simp*]: $(x^\star \cdot y^\star)^\star = x^\star \cdot y^\star$
by (*metis conway-c3 star-invol*)

lemma *conway-c2-var*: $(x \cdot y)^\star \cdot x \cdot y \cdot y^\star \leq (x \cdot y)^\star \cdot y^\star$
by (*metis mult-isor star-1r mult-assoc*)

lemma *conway-c2* [*simp*]: $(x \cdot y)^\star \cdot (x^\star + y^\star) = x^\star \cdot y^\star$
proof (*rule antisym*)
show $(x \cdot y)^\star \cdot (x^\star + y^\star) \leq x^\star \cdot y^\star$
by (*metis cka-1 conway-c3 prod-star-closure star-ext star-sum-var*)
have $x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) = x \cdot (x \cdot y)^\star \cdot (x^\star + 1 + y \cdot y^\star)$
by (*simp add*: *add-assoc*)
also have ... $= x \cdot (x \cdot y)^\star \cdot (x^\star + y \cdot y^\star)$
by (*simp add*: *add-commute*)
also have ... $= (x \cdot y)^\star \cdot (x \cdot x^\star) + (x \cdot y)^\star \cdot x \cdot y \cdot y^\star$
using *distrib-left mult-comm mult-assoc* **by** *force*
also have ... $\leq (x \cdot y)^\star \cdot x^\star + (x \cdot y)^\star \cdot x \cdot y \cdot y^\star$
using *add-iso mult-isol* **by** *force*
also have ... $\leq (x \cdot y)^\star \cdot x^\star + (x \cdot y)^\star \cdot y^\star$
using *conway-c2-var join.sup-mono* **by** *blast*
also have ... $= (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*simp add*: *distrib-left*)

finally have $x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$ **.**
moreover have $y^\star \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*metis dual-order.trans join.sup-ge2 mult-1-left mult-isor star-ref*)
ultimately have $y^\star + x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by *simp*
thus $x^\star \cdot y^\star \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*simp add*: *mult.assoc star-inductl*)
qed

end

end

1.14 Models of Dioids

theory *Dioid-Models*
imports *Kleene-Algebra.Dioid HOL.Real*
begin

In this section we consider some well known models of dioids. These so far include the powerset dioid over a monoid, languages, binary relations, sets of traces, sets paths (in a graph), as well as the min-plus and the max-plus semirings. Most of these models are taken from an article about Kleene algebras with domain [**?**].

The advantage of formally linking these models with the abstract axiomatisations of dioids is that all abstract theorems are automatically available in all models. It therefore makes sense to establish models for the strongest possible axiomatisations (whereas theorems should be proved for the weakest ones).

1.14.1 The Powerset Dioid over a Monoid

We assume a multiplicative monoid and define the usual complex product on sets of elements. We formalise the well known result that this lifting induces a dioid.

1.14.2 Language Dioids

Language dioids arise as special cases of the monoidal lifting because sets of words form free monoids. Moreover, monoids of words are isomorphic to monoids of lists under append.

To show that languages form dioids it therefore suffices to show that sets of lists closed under append and multiplication with the empty word form a (multiplicative) monoid. Isabelle then does the rest of the work automatically. Infix @ denotes word concatenation.

instantiation *list* :: (*type*) *monoid-mult*
begin

definition *times-list-def*:
$xs * ys \equiv xs\ @\ ys$

definition *one-list-def*:
$1 \equiv []$

instance proof
fix *xs ys zs* :: *'a list*
show $xs * ys * zs = xs * (ys * zs)$
by (*simp add*: *times-list-def*)
show $1 * xs = xs$
by (*simp add*: *one-list-def times-list-def*)
show $xs * 1 = xs$
by (*simp add*: *one-list-def times-list-def*)
qed

end

1.14.3 Relation Dioids

We now show that binary relations under union, relational composition, the identity relation, the empty relation and set inclusion form dioids. Due to the well developed relation library of Isabelle this is entirely trivial.

interpretation *rel-dioid*: *dioid-one-zero* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$
by (*unfold-locales*, *auto*)

interpretation *rel-monoid*: *monoid-mult Id* (O) **..**

1.14.4 Trace Dioids

Traces have been considered, for instance, by Kozen [**?**] in the context of Kleene algebras with tests. Intuitively, a trace is an execution sequence of a labelled transition system from some state to some other state, in which state labels and action labels alternate, and which begin and end with a state label.

Traces generalise words: words can be obtained from traces by forgetting state labels. Similarly, sets of traces generalise languages.

In this section we show that sets of traces under union, an appropriately defined notion of complex product, the set of all traces of length zero, the empty set of traces and set inclusion form a dioid.

We first define the notion of trace and the product of traces, which has been called *fusion product* by Kozen.

type-synonym (*'p*, *'a*) *trace* = $'p \times ('a \times 'p)$ *list*

definition *first* :: ('p, 'a) *trace* ⇒ 'p **where**
  first = *fst*

lemma *first-conv* [*simp*]: *first* (*p*, *xs*) = *p*
  by (*unfold first-def*, *simp*)

fun *last* :: ('p, 'a) *trace* ⇒ 'p **where**
  last (*p*, []) = *p*
| *last* (-, *xs*) = *snd* (*List.last xs*)

lemma *last-append* [*simp*]: *last* (*p*, *xs* @ *ys*) = *last* (*last* (*p*, *xs*), *ys*)
proof (*cases xs*)
  show *xs* = [] ⟹ *last* (*p*, *xs* @ *ys*) = *last* (*last* (*p*, *xs*), *ys*)
    by *simp*
  show ⋀*a list*. *xs* = *a* # *list* ⟹
    last (*p*, *xs* @ *ys*) = *last* (*last* (*p*, *xs*), *ys*)
  proof (*cases ys*)
    show ⋀*a list*. ⟦*xs* = *a* # *list*; *ys* = []⟧
      ⟹ *last* (*p*, *xs* @ *ys*) = *last* (*last* (*p*, *xs*), *ys*)
      by *simp*
    show ⋀*a list aa lista*. ⟦*xs* = *a* # *list*; *ys* = *aa* # *lista*⟧
      ⟹ *last* (*p*, *xs* @ *ys*) = *last* (*last* (*p*, *xs*), *ys*)
      by *simp*
  qed
qed

The fusion product is a partial operation. It is undefined if the last element of the first trace and the first element of the second trace are different. If these elements are the same, then the fusion product removes the first element from the second trace and appends the resulting object to the first trace.

definition *t-fusion* :: ('p, 'a) *trace* ⇒ ('p, 'a) *trace* ⇒ ('p, 'a) *trace* **where**
  t-fusion x y ≡ *if last x* = *first y then* (*fst x*, *snd x* @ *snd y*) *else undefined*

We now show that the first element and the last element of a trace are a left and right unit for that trace and prove some other auxiliary lemmas.

lemma *t-fusion-leftneutral* [*simp*]: *t-fusion* (*first x*, []) *x* = *x*
  by (*cases x*, *simp add*: *t-fusion-def*)

lemma *fusion-rightneutral* [*simp*]: *t-fusion x* (*last x*, []) = *x*
  by (*simp add*: *t-fusion-def*)

lemma *first-t-fusion* [*simp*]: *last x* = *first y* ⟹ *first* (*t-fusion x y*) = *first x*
  by (*simp add*: *first-def t-fusion-def*)

lemma *last-t-fusion* [*simp*]: *last x* = *first y* ⟹ *last* (*t-fusion x y*) = *last y*
  by (*simp add*: *first-def t-fusion-def*)

Next we show that fusion of traces is associative.

lemma *t-fusion-assoc* [*simp*]:
$[\![$ *last* x = *first* y; *last* y = *first* z $]\!] \Longrightarrow$ *t-fusion* x (*t-fusion* y z) = *t-fusion* (*t-fusion* x y) z
by (*cases* x, *cases* y, *cases* z, *simp add*: *t-fusion-def*)

1.14.5 Sets of Traces

We now lift the fusion product to a complex product on sets of traces. This operation is total.

no-notation
times (**infixl** $\cdot$ *70*)

definition *t-prod* :: ('*p*, '*a*) *trace set* $\Rightarrow$ ('*p*, '*a*) *trace set* $\Rightarrow$ ('*p*, '*a*) *trace set* (**infixl** $\cdot$ *70*)
where $X \cdot Y = \{$*t-fusion* u v| u v. $u \in X \wedge v \in Y \wedge$ *last* u = *first* $v\}$

Next we define the empty set of traces and the set of traces of length zero as the multiplicative unit of the trace dioid.

definition *t-zero* :: ('*p*, '*a*) *trace set* **where**
t-zero $\equiv \{\}$

definition *t-one* :: ('*p*, '*a*) *trace set* **where**
t-one $\equiv \bigcup p.\ \{(p, [])\}$

We now provide elimination rules for trace products.

lemma *t-prod-iff*:
$w \in X \cdot Y \longleftrightarrow (\exists\, u\ v.\ w =$ *t-fusion* u $v \wedge u \in X \wedge v \in Y \wedge$ *last* u = *first* v)
by (*unfold t-prod-def*) *auto*

lemma *t-prod-intro* [*simp*, *intro*]:
$[\![\ u \in X;\ v \in Y;$ *last* u = *first* v $]\!] \Longrightarrow$ *t-fusion* u $v \in X \cdot Y$
by (*meson t-prod-iff*)

lemma *t-prod-elim* [*elim*]:
$w \in X \cdot Y \Longrightarrow \exists\, u\ v.\ w =$ *t-fusion* u $v \wedge u \in X \wedge v \in Y \wedge$ *last* u = *first* v
by (*meson t-prod-iff*)

Finally we prove the interpretation statement that sets of traces under union and the complex product based on trace fusion together with the empty set of traces and the set of traces of length one forms a dioid.

interpretation *trace-dioid*: *dioid-one-zero* ($\cup$) *t-prod t-one t-zero* ($\subseteq$) ($\subset$)
apply *unfold-locales*
apply (*auto simp add*: *t-prod-def t-one-def t-zero-def t-fusion-def*)
apply (*metis last-append*)
apply (*metis last-append append-assoc*)
done

```
no-notation
  t-prod (infixl · 70)
```

1.14.6 The Path Diod

The next model we consider are sets of paths in a graph. We consider two variants, one that contains the empty path and one that doesn't. The former leads to more difficult proofs and a more involved specification of the complex product. We start with paths that include the empty path. In this setting, a path is a list of nodes.

1.14.7 Path Models with the Empty Path

```
type-synonym 'a path = 'a list
```

Path fusion is defined similarly to trace fusion. Mathematically it should be a partial operation. The fusion of two empty paths yields the empty path; the fusion between a non-empty path and an empty one is undefined; the fusion of two non-empty paths appends the tail of the second path to the first one.

We need to use a total alternative and make sure that undefined paths do not contribute to the complex product.

```
fun p-fusion :: 'a path ⇒ 'a path ⇒ 'a path where
  p-fusion [] - = []
| p-fusion - [] = []
| p-fusion ps (q # qs) = ps @ qs

lemma p-fusion-assoc:
  p-fusion ps (p-fusion qs rs) = p-fusion (p-fusion ps qs) rs
proof (induct rs)
  case Nil show ?case
    by (metis p-fusion.elims p-fusion.simps(2))
  case Cons show ?case
  proof (induct qs)
    case Nil show ?case
      by (metis neq-Nil-conv p-fusion.simps(1) p-fusion.simps(2))
    case Cons show ?case
      proof –
        have ∀ ps. ([] = ps ∨ hd ps # tl ps = ps) ∧ ((∀ q qs. q # qs ≠ ps) ∨ [] ≠
ps)
          using list.collapse by fastforce
        moreover hence ∀ ps q qs. p-fusion ps (q # qs) = ps @ qs ∨ [] = ps
          by (metis p-fusion.simps(3))
        ultimately show ?thesis
         by (metis (no-types) Cons-eq-appendI append-eq-appendI p-fusion.simps(1)
p-fusion.simps(3))
```

qed
qed
qed

This lemma overapproximates the real situation, but it holds in all cases where path fusion should be defined.

lemma *p-fusion-last*:
assumes *List.last ps* $=$ *hd qs*
and *ps* $\neq$ []
and *qs* $\neq$ []
shows *List.last* (*p-fusion ps qs*) $=$ *List.last qs*
by (*metis* (*hide-lams*, *no-types*) *List.last.simps List.last-append append-Nil2 assms list.sel*(*1*) *neq-Nil-conv p-fusion.simps*(*3*))

lemma *p-fusion-hd*: $\llbracket$*ps* $\neq$ []; *qs* $\neq$ []$\rrbracket \Longrightarrow$ *hd* (*p-fusion ps qs*) $=$ *hd ps*
by (*metis list.exhaust p-fusion.simps*(*3*) *append-Cons list.sel*(*1*))

lemma *nonempty-p-fusion*: $\llbracket$*ps* $\neq$ []; *qs* $\neq$ []$\rrbracket \Longrightarrow$ *p-fusion ps qs* $\neq$ []
by (*metis list.exhaust append-Cons p-fusion.simps*(*3*) *list.simps*(*2*))

We now define a condition that filters out undefined paths in the complex product.

abbreviation *p-filter* :: *'a path* $\Rightarrow$ *'a path* $\Rightarrow$ *bool* **where**
p-filter ps qs $\equiv$ ((*ps* $=$ [] $\wedge$ *qs* $=$ []) $\vee$ (*ps* $\neq$ [] $\wedge$ *qs* $\neq$ [] $\wedge$ (*List.last ps*) $=$ *hd qs*))

no-notation
times (**infixl** $\cdot$ *70*)

definition *p-prod* :: *'a path set* $\Rightarrow$ *'a path set* $\Rightarrow$ *'a path set* (**infixl** $\cdot$ *70*)
where $X \cdot Y = \{$*rs* . $\exists$ *ps* $\in X$. $\exists$ *qs* $\in Y$. *rs* $=$ *p-fusion ps qs* $\wedge$ *p-filter ps qs*$\}$

lemma *p-prod-iff*:
ps $\in X \cdot Y \longleftrightarrow$ ($\exists$ *qs rs*. *ps* $=$ *p-fusion qs rs* $\wedge$ *qs* $\in X \wedge$ *rs* $\in Y \wedge$ *p-filter qs rs*)
by (*unfold p-prod-def*) *auto*

Due to the complexity of the filter condition, proving properties of complex products can be tedious.

lemma *p-prod-assoc*: $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
proof (*rule set-eqI*)
fix *ps*
show *ps* $\in (X \cdot Y) \cdot Z \longleftrightarrow$ *ps* $\in X \cdot (Y \cdot Z)$
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by *auto* (*metis nonempty-p-fusion p-prod-iff*)+
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-prod-iff*) (*metis* (*hide-lams*, *mono-tags*) *nonempty-p-fusion p-fusion-assoc p-fusion-hd p-fusion-last*)+

qed
qed

We now define the multiplicative unit of the path dioid as the set of all paths of length one, including the empty path, and show the unit laws with respect to the path product.

definition *p-one* :: *'a path set* **where**
p-one ≡ {*p* . ∃ *q*::*'a*. *p* = [*q*]} ∪ {[]}

lemma *p-prod-onel* [*simp*]: *p-one* · *X* = *X*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *p-one* · *X* ⟷ *ps* ∈ *X*
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis nonempty-p-fusion not-Cons-self*)
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis append-Cons append-Nil list.sel*(*1*) *neq-Nil-conv p-fusion.simps*(*3*), *metis Cons-eq-appendI list.sel*(*1*) *last-ConsL list.simps*(*3*) *p-fusion.simps*(*3*) *self-append-conv2*)
qed
qed

lemma *p-prod-oner* [*simp*]: *X* · *p-one* = *X*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *X* · *p-one* ⟷ *ps* ∈ *X*
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis nonempty-p-fusion not-Cons-self2*, *metis p-fusion.simps*(*1*))
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis append-Nil2 neq-Nil-conv p-fusion.simps*(*3*), *metis list.sel*(*1*) *list.simps*(*2*) *p-fusion.simps*(*3*) *self-append-conv*)
qed
qed

Next we show distributivity laws at the powerset level.

lemma *p-prod-distl*: *X* · (*Y* ∪ *Z*) = *X* · *Y* ∪ *X* · *Z*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *X* · (*Y* ∪ *Z*) ⟷ *ps* ∈ *X* · *Y* ∪ *X* · *Z*
by (*cases ps*) (*auto simp add*: *p-prod-iff*)
qed

lemma *p-prod-distr*: (*X* ∪ *Y*) · *Z* = *X* · *Z* ∪ *Y* · *Z*

proof (*rule set-eqI*)
  fix *ps*
  show $ps \in (X \cup Y) \cdot Z \longleftrightarrow ps \in X \cdot Z \cup Y \cdot Z$
    by (*cases ps*) (*auto simp add*: *p-prod-iff*)
qed

Finally we show that sets of paths under union, the complex product, the unit set and the empty set form a dioid.

interpretation *path-dioid*: *dioid-one-zero* $(\cup)$ $(\cdot)$ *p-one* $\{\}$ $(\subseteq)$ $(\subset)$
proof
  fix $x\ y\ z$ :: *'a path set*
  show $x \cup y \cup z = x \cup (y \cup z)$
    by *auto*
  show $x \cup y = y \cup x$
    by *auto*
  show $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
    by (*fact p-prod-assoc*)
  show $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$
    by (*fact p-prod-distr*)
  show *p-one* $\cdot\ x = x$
    by (*fact p-prod-onel*)
  show $x \cdot$ *p-one* $= x$
    by (*fact p-prod-oner*)
  show $\{\} \cup x = x$
    by *auto*
  show $\{\} \cdot x = \{\}$
    by (*metis all-not-in-conv p-prod-iff*)
  show $x \cdot \{\} = \{\}$
    by (*metis all-not-in-conv p-prod-iff*)
  show $(x \subseteq y) = (x \cup y = y)$
    by *auto*
  show $(x \subset y) = (x \subseteq y \wedge x \neq y)$
    by *auto*
  show $x \cup x = x$
    by *auto*
  show $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$
    by (*fact p-prod-distl*)
qed

no-notation
  p-prod (**infixl** $\cdot$ *70*)

1.14.8 Path Models without the Empty Path

We now build a model of paths that does not include the empty path and therefore leads to a simpler complex product.

datatype *'a ppath* = *Node 'a* | *Cons 'a 'a ppath*

primrec *pp-first* :: $'a$ *ppath* $\Rightarrow$ $'a$ **where**
pp-first $(Node\ x) = x$
| *pp-first* $(Cons\ x\ \text{-}) = x$

primrec *pp-last* :: $'a$ *ppath* $\Rightarrow$ $'a$ **where**
pp-last $(Node\ x) = x$
| *pp-last* $(Cons\ \text{-}\ xs) =$ *pp-last xs*

The path fusion product (although we define it as a total funcion) should only be applied when the last element of the first argument is equal to the first element of the second argument.

primrec *pp-fusion* :: $'a$ *ppath* $\Rightarrow$ $'a$ *ppath* $\Rightarrow$ $'a$ *ppath* **where**
pp-fusion $(Node\ x)\ ys = ys$
| *pp-fusion* $(Cons\ x\ xs)\ ys = Cons\ x$ (*pp-fusion xs ys*)

We now go through the same steps as for traces and paths before, showing that the first and last element of a trace a left or right unit for that trace and that the fusion product on traces is associative.

lemma *pp-fusion-leftneutral* [*simp*]: *pp-fusion* (*Node* (*pp-first* x)) $x = x$
by *simp*

lemma *pp-fusion-rightneutral* [*simp*]: *pp-fusion* x (*Node* (*pp-last* x)) $= x$
by (*induct* x) *simp-all*

lemma *pp-first-pp-fusion* [*simp*]:
pp-last $x =$ *pp-first* $y \Longrightarrow$ *pp-first* (*pp-fusion* $x\ y$) $=$ *pp-first* x
by (*induct* x) *simp-all*

lemma *pp-last-pp-fusion* [*simp*]:
pp-last $x =$ *pp-first* $y \Longrightarrow$ *pp-last* (*pp-fusion* $x\ y$) $=$ *pp-last* y
by (*induct* x) *simp-all*

lemma *pp-fusion-assoc* [*simp*]:
$[\![$ *pp-last* $x =$ *pp-first* y; *pp-last* $y =$ *pp-first* z $]\!] \Longrightarrow$ *pp-fusion* x (*pp-fusion* $y\ z$) $=$ *pp-fusion* (*pp-fusion* $x\ y$) z
by (*induct* x) *simp-all*

We now lift the path fusion product to a complex product on sets of paths. This operation is total.

definition *pp-prod* :: $'a$ *ppath set* $\Rightarrow$ $'a$ *ppath set* $\Rightarrow$ $'a$ *ppath set* (**infixl** $\cdot$ *70*)
where $X \cdot Y = \{$*pp-fusion* $u\ v|\ u\ v.\ u \in X \wedge v \in Y \wedge$ *pp-last* $u =$ *pp-first* $v\}$

Next we define the set of paths of length one as the multiplicative unit of the path dioid.

definition *pp-one* :: $'a$ *ppath set* **where**
pp-one $\equiv$ *range Node*

We again provide an elimination rule.

lemma *pp-prod-iff*:
  $w \in X \cdot Y \longleftrightarrow (\exists\, u\ v.\ w = \textit{pp-fusion}\ u\ v \wedge u \in X \wedge v \in Y \wedge \textit{pp-last}\ u = \textit{pp-first}\ v)$
  by (*unfold pp-prod-def*) *auto*

interpretation *ppath-dioid*: *dioid-one-zero* $(\cup)$ $(\cdot)$ *pp-one* $\{\}$ $(\subseteq)$ $(\subset)$
proof
  fix $x\ y\ z$ :: *'a ppath set*
  show $x \cup y \cup z = x \cup (y \cup z)$
    by *auto*
  show $x \cup y = y \cup x$
    by *auto*
  show $x \cdot y \cdot z = x \cdot (y \cdot z)$
    by (*auto simp add*: *pp-prod-def*, *metis pp-first-pp-fusion pp-fusion-assoc*, *metis pp-last-pp-fusion*)
  show $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$
    by (*auto simp add*: *pp-prod-def*)
  show *pp-one* $\cdot\ x = x$
    by (*auto simp add*: *pp-one-def pp-prod-def*, *metis pp-fusion.simps*(*1*) *pp-last.simps*(*1*) *rangeI*)
  show $x \cdot$ *pp-one* $= x$
    by (*auto simp add*: *pp-one-def pp-prod-def*, *metis pp-first.simps*(*1*) *pp-fusion-rightneutral rangeI*)
  show $\{\} \cup x = x$
    by *auto*
  show $\{\} \cdot x = \{\}$
    by (*simp add*: *pp-prod-def*)
  show $x \cdot \{\} = \{\}$
    by (*simp add*: *pp-prod-def*)
  show $x \subseteq y \longleftrightarrow x \cup y = y$
    by *auto*
  show $x \subset y \longleftrightarrow x \subseteq y \wedge x \neq y$
    by *auto*
  show $x \cup x = x$
    by *auto*
  show $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$
    by (*auto simp add*: *pp-prod-def*)
qed

no-notation
  pp-prod (**infixl** $\cdot$ *70*)

1.14.9 The Distributive Lattice Dioid

A bounded distributive lattice is a distributive lattice with a least and a greatest element. Using Isabelle's lattice theory file we define a bounded distributive lattice as an axiomatic type class and show, using a sublocale statement, that every bounded distributive lattice is a dioid with one and zero.

class *bounded-distributive-lattice* = *bounded-lattice* + *distrib-lattice*

sublocale *bounded-distributive-lattice* ⊆ *dioid-one-zero sup inf top bot less-eq*
proof
  fix $x\ y\ z$
  show $sup\ (sup\ x\ y)\ z = sup\ x\ (sup\ y\ z)$
    by (*fact sup-assoc*)
  show $sup\ x\ y = sup\ y\ x$
    by (*fact sup.commute*)
  show $inf\ (inf\ x\ y)\ z = inf\ x\ (inf\ y\ z)$
    by (*metis inf.commute inf.left-commute*)
  show $inf\ (sup\ x\ y)\ z = sup\ (inf\ x\ z)\ (inf\ y\ z)$
    by (*fact inf-sup-distrib2*)
  show $inf\ top\ x = x$
    by *simp*
  show $inf\ x\ top = x$
    by *simp*
  show $sup\ bot\ x = x$
    by *simp*
  show $inf\ bot\ x = bot$
    by *simp*
  show $inf\ x\ bot = bot$
    by *simp*
  show $(x \leq y) = (sup\ x\ y = y)$
    by (*fact le-iff-sup*)
  show $(x < y) = (x \leq y \wedge x \neq y)$
    by *auto*
  show $sup\ x\ x = x$
    by *simp*
  show $inf\ x\ (sup\ y\ z) = sup\ (inf\ x\ y)\ (inf\ x\ z)$
    by (*fact inf-sup-distrib1*)
qed

1.14.10 The Boolean Dioid

In this section we show that the booleans form a dioid, because the booleans form a bounded distributive lattice.

instantiation *bool* :: *bounded-distributive-lattice*
begin

  instance ..

end

interpretation *boolean-dioid*: *dioid-one-zero sup inf True False less-eq less*
  by (*unfold-locales*, *simp-all add*: *inf-bool-def sup-bool-def*)

1.14.11 The Max-Plus Dioid

The following dioids have important applications in combinatorial optimisations, control theory, algorithm design and computer networks.

A definition of reals extended with $+\infty$ *and* $-\infty$ may be found in *HOL/Library/Extended_Real.thy*. Alas, we require separate extensions with either $+\infty$ or $-\infty$.

The carrier set of the max-plus semiring is the set of real numbers extended by minus infinity. The operation of addition is maximum, the operation of multiplication is addition, the additive unit is minus infinity and the multiplicative unit is zero.

datatype *mreal* = *mreal real* | *MInfty* — minus infinity

fun *mreal-max* **where**
  mreal-max (*mreal x*) (*mreal y*) = *mreal* (*max x y*)
| *mreal-max x MInfty* = *x*
| *mreal-max MInfty y* = *y*

lemma *mreal-max-simp-3* [*simp*]: *mreal-max MInfty y* = *y*
  by (*cases y*, *simp-all*)

fun *mreal-plus* **where**
  mreal-plus (*mreal x*) (*mreal y*) = *mreal* (*x* + *y*)
| *mreal-plus* - - = *MInfty*

We now show that the max plus-semiring satisfies the axioms of selective semirings, from which it follows that it satisfies the dioid axioms.

instantiation *mreal* :: *selective-semiring*
begin

  definition *zero-mreal-def*:
    0 ≡ *MInfty*

  definition *one-mreal-def*:
    1 ≡ *mreal 0*

  definition *plus-mreal-def*:
    x + *y* ≡ *mreal-max x y*

  definition *times-mreal-def*:
    x ∗ *y* ≡ *mreal-plus x y*

  definition *less-eq-mreal-def*:
    (*x*::*mreal*) ≤ *y* ≡ *x* + *y* = *y*

  definition *less-mreal-def*:
    (*x*::*mreal*) < *y* ≡ *x* ≤ *y* ∧ *x* ≠ *y*

instance
proof
fix *x y z* :: *mreal*
show $x + y + z = x + (y + z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-mreal-def*)
show $x + y = y + x$
by (*cases x*, *cases y*, *simp-all add*: *plus-mreal-def*)
show $x * y * z = x * (y * z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *times-mreal-def*)
show $(x + y) * z = x * z + y * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-mreal-def times-mreal-def*)
show $\mathit{1} * x = x$
by (*cases x*, *simp-all add*: *one-mreal-def times-mreal-def*)
show $x * \mathit{1} = x$
by (*cases x*, *simp-all add*: *one-mreal-def times-mreal-def*)
show $\mathit{0} + x = x$
by (*cases x*, *simp-all add*: *plus-mreal-def zero-mreal-def*)
show $\mathit{0} * x = \mathit{0}$
by (*cases x*, *simp-all add*: *times-mreal-def zero-mreal-def*)
show $x * \mathit{0} = \mathit{0}$
by (*cases x*, *simp-all add*: *times-mreal-def zero-mreal-def*)
show $x \leq y \longleftrightarrow x + y = y$
by (*metis less-eq-mreal-def*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$
by (*metis less-mreal-def*)
show $x + y = x \vee x + y = y$
by (*cases x*, *cases y*, *simp-all add*: *plus-mreal-def*, *metis linorder-le-cases max.absorb-iff2 max.absorb1*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-mreal-def times-mreal-def*)
qed

end

1.14.12 The Min-Plus Dioid

The min-plus dioid is also known as *tropical semiring*. Here we need to add a positive infinity to the real numbers. The procedere follows that of max-plus semirings.

datatype *preal* = *preal real* | *PInfty* — plus infinity

fun *preal-min* **where**
preal-min (*preal x*) (*preal y*) = *preal* (*min x y*)
| *preal-min x PInfty* = *x*
| *preal-min PInfty y* = *y*

lemma *preal-min-simp-3* [*simp*]: *preal-min PInfty y* = *y*
by (*cases y*, *simp-all*)

```
fun preal-plus where
  preal-plus (preal x) (preal y) = preal (x + y)
| preal-plus - - = PInfty

instantiation preal :: selective-semiring
begin

  definition zero-preal-def:
    0 ≡ PInfty

  definition one-preal-def:
    1 ≡ preal 0

  definition plus-preal-def:
    x + y ≡ preal-min x y

  definition times-preal-def:
    x ∗ y ≡ preal-plus x y

  definition less-eq-preal-def:
    (x::preal) ≤ y ≡ x + y = y

  definition less-preal-def:
    (x::preal) < y ≡ x ≤ y ∧ x ≠ y

  instance
  proof
    fix x y z :: preal
    show x + y + z = x + (y + z)
      by (cases x, cases y, cases z, simp-all add: plus-preal-def)
    show x + y = y + x
      by (cases x, cases y, simp-all add: plus-preal-def)
    show x ∗ y ∗ z = x ∗ (y ∗ z)
      by (cases x, cases y, cases z, simp-all add: times-preal-def)
    show (x + y) ∗ z = x ∗ z + y ∗ z
      by (cases x, cases y, cases z, simp-all add: plus-preal-def times-preal-def)
    show 1 ∗ x = x
      by (cases x, simp-all add: one-preal-def times-preal-def)
    show x ∗ 1 = x
      by (cases x, simp-all add: one-preal-def times-preal-def)
    show 0 + x = x
      by (cases x, simp-all add: plus-preal-def zero-preal-def)
    show 0 ∗ x = 0
      by (cases x, simp-all add: times-preal-def zero-preal-def)
    show x ∗ 0 = 0
      by (cases x, simp-all add: times-preal-def zero-preal-def)
    show x ≤ y ⟷ x + y = y
      by (metis less-eq-preal-def)
    show x < y ⟷ x ≤ y ∧ x ≠ y
```

by (*metis less-preal-def*)
show $x + y = x \vee x + y = y$
by (*cases x*, *cases y*, *simp-all add*: *plus-preal-def*, *metis linorder-le-cases min.absorb2 min.absorb-iff1*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-preal-def times-preal-def*)
qed

end

Variants of min-plus and max-plus semirings can easily be obtained. Here we formalise the min-plus semiring over the natural numbers as an example.

datatype *pnat* = *pnat nat* | *PInfty* — plus infinity

fun *pnat-min* **where**
pnat-min (*pnat x*) (*pnat y*) = *pnat* (*min x y*)
| *pnat-min x PInfty* = *x*
| *pnat-min PInfty x* = *x*

lemma *pnat-min-simp-3* [*simp*]: *pnat-min PInfty y* = *y*
by (*cases y*, *simp-all*)

fun *pnat-plus* **where**
pnat-plus (*pnat x*) (*pnat y*) = *pnat* ($x + y$)
| *pnat-plus - -* = *PInfty*

instantiation *pnat* :: *selective-semiring*
begin

definition *zero-pnat-def*:
0 ≡ *PInfty*

definition *one-pnat-def*:
1 ≡ *pnat 0*

definition *plus-pnat-def*:
$x + y$ ≡ *pnat-min x y*

definition *times-pnat-def*:
$x * y$ ≡ *pnat-plus x y*

definition *less-eq-pnat-def*:
(*x*::*pnat*) $\le y \equiv x + y = y$

definition *less-pnat-def*:
(*x*::*pnat*) $< y \equiv x \le y \wedge x \neq y$

lemma *zero-pnat-top*: (*x*::*pnat*) $\le$ *1*
by (*cases x*, *simp-all add*: *less-eq-pnat-def plus-pnat-def one-pnat-def*)

instance
proof
fix $x\ y\ z$ :: *pnat*
show $x + y + z = x + (y + z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def*)
show $x + y = y + x$
by (*cases x*, *cases y*, *simp-all add*: *plus-pnat-def*)
show $x * y * z = x * (y * z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *times-pnat-def*)
show $(x + y) * z = x * z + y * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def times-pnat-def*)
show $1 * x = x$
by (*cases x*, *simp-all add*: *one-pnat-def times-pnat-def*)
show $x * 1 = x$
by (*cases x*, *simp-all add*: *one-pnat-def times-pnat-def*)
show $0 + x = x$
by (*cases x*, *simp-all add*: *plus-pnat-def zero-pnat-def*)
show $0 * x = 0$
by (*cases x*, *simp-all add*: *times-pnat-def zero-pnat-def*)
show $x * 0 = 0$
by (*cases x*, *simp-all add*: *times-pnat-def zero-pnat-def*)
show $x \leq y \longleftrightarrow x + y = y$
by (*metis less-eq-pnat-def*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$
by (*metis less-pnat-def*)
show $x + y = x \vee x + y = y$
by (*cases x*, *cases y*, *simp-all add*: *plus-pnat-def*, *metis linorder-le-cases min.absorb2 min.absorb-iff1*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def times-pnat-def*)
qed

end

end

1.15 Models of Kleene Algebras

theory *Kleene-Algebra-Models*
imports *Kleene-Algebra Dioid-Models*
begin

We now show that most of the models considered for dioids are also Kleene algebras. Some of the dioid models cannot be expanded, for instance max-plus and min-plus semirings, but we do not formalise this fact. We also currently do not show that formal powerseries and matrices form Kleene algebras.

The interpretation proofs for some of the following models are quite similar. One could, perhaps, abstract out common reasoning in the future.

1.15.1 Preliminary Lemmas

We first prove two induction-style statements for dioids that are useful for establishing the full induction laws. In the future these will live in a theory file on finite sums for Kleene algebras.

context *dioid-one-zero*
begin

lemma *power-inductl*: $z + x \cdot y \leq y \Longrightarrow (x \mathbin{\hat{}} n) \cdot z \leq y$
proof (*induct n*)
  case *0* **show** *?case*
    using *0.prems* **by** *auto*
  case *Suc* **thus** *?case*
    by (*auto*, *metis mult.assoc mult-isol order-trans*)
qed

lemma *power-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot (x \mathbin{\hat{}} n) \leq y$
proof (*induct n*)
  case *0* **show** *?case*
    using *0.prems* **by** *auto*
  case *Suc*
  {
    fix *n*
    assume $z + y \cdot x \leq y \Longrightarrow z \cdot x \mathbin{\hat{}} n \leq y$
      and $z + y \cdot x \leq y$
    hence $z \cdot x \mathbin{\hat{}} n \leq y$
      by *auto*
    also have $z \cdot x \mathbin{\hat{}} \mathit{Suc}\ n = z \cdot x \cdot x \mathbin{\hat{}} n$
      by (*metis mult.assoc power-Suc*)
    moreover have ... $= (z \cdot x \mathbin{\hat{}} n) \cdot x$
      by (*metis mult.assoc power-commutes*)
    moreover have ... $\leq y \cdot x$
      by (*metis calculation*(*1*) *mult-isor*)
    moreover have ... $\leq y$
      using $\langle z + y \cdot x \leq y \rangle$ **by** *auto*
    ultimately have $z \cdot x \mathbin{\hat{}} \mathit{Suc}\ n \leq y$ **by** *auto*
  }
  thus *?case*
    by (*metis Suc*)
qed

end

1.15.2 The Powerset Kleene Algebra over a Monoid

We now show that the powerset dioid forms a Kleene algebra. The Kleene star is defined as in language theory.

lemma *Un-0-Suc*: $(\bigcup n. f\ n) = f\ 0 \cup (\bigcup n. f\ (Suc\ n))$
by *auto* (*metis not0-implies-Suc*)

1.15.3 Relation Kleene Algebras

We now show that binary relations form Kleene algebras. While we could have used the reflexive transitive closure operation as the Kleene star, we prefer the equivalent definition of the star as the sum of powers. This essentially allows us to copy previous proofs.

lemma *power-is-relpow*: *rel-dioid.power* $X\ n = X$ ^^ n
proof (*induct n*)
  case *0* **show** *?case*
    by (*metis rel-dioid.power-0 relpow.simps*(*1*))
  case *Suc* **thus** *?case*
    by (*metis rel-dioid.power-Suc2 relpow.simps*(*2*))
qed

lemma *rel-star-def*: X^$* = (\bigcup n.$ *rel-dioid.power* $X\ n)$
  by (*simp add*: *power-is-relpow rtrancl-is-UN-relpow*)

lemma *rel-star-contl*: $X\ O\ Y$^$* = (\bigcup n.\ X\ O$ *rel-dioid.power* $Y\ n)$
by (*metis rel-star-def relcomp-UNION-distrib*)

lemma *rel-star-contr*: X^$*\ O\ Y = (\bigcup n.$ (*rel-dioid.power* $X\ n)\ O\ Y)$
by (*metis rel-star-def relcomp-UNION-distrib2*)

interpretation *rel-kleene-algebra*: *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-dioid.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-dioid.power-inductr*)
qed

1.15.4 Trace Kleene Algebras

Again, the proof that sets of traces form Kleene algebras follows the same schema.

definition *t-star* :: ('*p*, '*a*) *trace set* ⇒ ('*p*, '*a*) *trace set* **where**
t-star X ≡ ⋃*n*. *trace-dioid.power X n*

lemma *t-star-elim*: *x* ∈ *t-star X* ⟷ (∃ *n*. *x* ∈ *trace-dioid.power X n*)
by (*simp add*: *t-star-def*)

lemma *t-star-contl*: *t-prod X* (*t-star Y*) = (⋃*n*. *t-prod X* (*trace-dioid.power Y n*))
by (*auto simp add*: *t-star-elim t-prod-def*)

lemma *t-star-contr*: *t-prod* (*t-star X*) *Y* = (⋃*n*. *t-prod* (*trace-dioid.power X n*) *Y*)
by (*auto simp add*: *t-star-elim t-prod-def*)

interpretation *trace-kleene-algebra*: *kleene-algebra* (∪) *t-prod t-one t-zero* (⊆) (⊂) *t-star*
proof
fix *X Y Z* :: ('*a*, '*b*) *trace set*
show *t-one* ∪ *t-prod X* (*t-star X*) ⊆ *t-star X*
proof −
have *t-one* ∪ *t-prod X* (*t-star X*) = (*trace-dioid.power X 0*) ∪ (⋃*n*. *trace-dioid.power X* (*Suc n*))
by (*auto simp add*: *t-star-def t-prod-def*)
also have ... = (⋃*n*. *trace-dioid.power X n*)
by (*metis Un-0-Suc*)
also have ... = *t-star X*
by (*metis t-star-def*)
finally show *?thesis*
by (*metis subset-refl*)
qed
show *Z* ∪ *t-prod X Y* ⊆ *Y* ⟹ *t-prod* (*t-star X*) *Z* ⊆ *Y*
by (*simp only*: *ball-UNIV t-star-contr SUP-le-iff*) (*metis trace-dioid.power-inductl*)
show *Z* ∪ *t-prod Y X* ⊆ *Y* ⟹ *t-prod Z* (*t-star X*) ⊆ *Y*
by (*simp only*: *ball-UNIV t-star-contl SUP-le-iff*) (*metis trace-dioid.power-inductr*)
qed

1.15.5 Path Kleene Algebras

We start with paths that include the empty path.

definition *p-star* :: '*a path set* ⇒ '*a path set* **where**
p-star X ≡ ⋃*n*. *path-dioid.power X n*

lemma *p-star-elim*: *x* ∈ *p-star X* ⟷ (∃ *n*. *x* ∈ *path-dioid.power X n*)
by (*simp add*: *p-star-def*)

lemma *p-star-contl*: *p-prod X* (*p-star Y*) = ($\bigcup$*n*. *p-prod X* (*path-dioid.power Y n*))
apply (*auto simp add*: *p-prod-def p-star-elim*)
  apply (*metis p-fusion.simps*(*1*))
 apply *metis*
 apply (*metis p-fusion.simps*(*1*) *p-star-elim*)
apply (*metis p-star-elim*)
done

lemma *p-star-contr*: *p-prod* (*p-star X*) *Y* = ($\bigcup$*n*. *p-prod* (*path-dioid.power X n*) *Y*)
apply (*auto simp add*: *p-prod-def p-star-elim*)
  apply (*metis p-fusion.simps*(*1*))
 apply *metis*
 apply (*metis p-fusion.simps*(*1*) *p-star-elim*)
apply (*metis p-star-elim*)
done

interpretation *path-kleene-algebra*: *kleene-algebra* (∪) *p-prod p-one* {} (⊆) (⊂) *p-star*
proof
 fix *X Y Z* :: *'a path set*
 show *p-one* ∪ *p-prod X* (*p-star X*) ⊆ *p-star X*
  proof −
   have *p-one* ∪ *p-prod X* (*p-star X*) = (*path-dioid.power X 0*) ∪ ($\bigcup$*n*. *path-dioid.power X* (*Suc n*))
    by (*auto simp add*: *p-star-def p-prod-def*)
   also have ... = ($\bigcup$*n*. *path-dioid.power X n*)
    by (*metis Un-0-Suc*)
   also have ... = *p-star X*
    by (*metis p-star-def*)
   finally show *?thesis*
    by (*metis subset-refl*)
  qed
 show *Z* ∪ *p-prod X Y* ⊆ *Y* ⟹ *p-prod* (*p-star X*) *Z* ⊆ *Y*
  by (*simp only*: *ball-UNIV p-star-contr SUP-le-iff*) (*metis path-dioid.power-inductl*)
 show *Z* ∪ *p-prod Y X* ⊆ *Y* ⟹ *p-prod Z* (*p-star X*) ⊆ *Y*
  by (*simp only*: *ball-UNIV p-star-contl SUP-le-iff*) (*metis path-dioid.power-inductr*)
qed

We now consider a notion of paths that does not include the empty path.

definition *pp-star* :: *'a ppath set* ⇒ *'a ppath set* **where**
 pp-star X ≡ $\bigcup$*n*. *ppath-dioid.power X n*

lemma *pp-star-elim*: *x* ∈ *pp-star X* ⟷ (∃ *n*. *x* ∈ *ppath-dioid.power X n*)
by (*simp add*: *pp-star-def*)

lemma *pp-star-contl*: *pp-prod X* (*pp-star Y*) = ($\bigcup$*n*. *pp-prod X* (*ppath-dioid.power*

```
Y n))
by (auto simp add: pp-prod-def pp-star-elim)

lemma pp-star-contr: pp-prod (pp-star X) Y = (⋃ n. pp-prod (ppath-dioid.power
X n) Y)
by (auto simp add: pp-prod-def pp-star-elim)

interpretation ppath-kleene-algebra: kleene-algebra (∪) pp-prod pp-one {} (⊆) (⊂)
pp-star
proof
  fix X Y Z :: 'a ppath set
  show pp-one ∪ pp-prod X (pp-star X) ⊆ pp-star X
    proof −
      have pp-one ∪ pp-prod X (pp-star X) = (ppath-dioid.power X 0) ∪ (⋃ n.
ppath-dioid.power X (Suc n))
        by (auto simp add: pp-star-def pp-prod-def)
      also have ... = (⋃ n. ppath-dioid.power X n)
        by (metis Un-0-Suc)
      also have ... = pp-star X
        by (metis pp-star-def)
      finally show ?thesis
        by (metis subset-refl)
    qed
  show Z ∪ pp-prod X Y ⊆ Y ⟹ pp-prod (pp-star X) Z ⊆ Y
   by (simp only: ball-UNIV pp-star-contr SUP-le-iff) (metis ppath-dioid.power-inductl)
  show Z ∪ pp-prod Y X ⊆ Y ⟹ pp-prod Z (pp-star X) ⊆ Y
   by (simp only: ball-UNIV pp-star-contl SUP-le-iff) (metis ppath-dioid.power-inductr)
qed
```

1.15.6 The Distributive Lattice Kleene Algebra

In the case of bounded distributive lattices, the star maps all elements to to the maximal element.

```
definition (in bounded-distributive-lattice) bdl-star :: 'a ⇒ 'a where
  bdl-star x = top

sublocale bounded-distributive-lattice ⊆ kleene-algebra sup inf top bot less-eq less
bdl-star
proof
  fix x y z :: 'a
  show sup top (inf x (bdl-star x)) ≤ bdl-star x
    by (simp add: bdl-star-def)
  show sup z (inf x y) ≤ y ⟹ inf (bdl-star x) z ≤ y
    by (simp add: bdl-star-def)
  show sup z (inf y x) ≤ y ⟹ inf z (bdl-star x) ≤ y
    by (simp add: bdl-star-def)
qed
```

1.15.7 The Min-Plus Kleene Algebra

One cannot define a Kleene star for max-plus and min-plus algebras that range over the real numbers. Here we define the star for a min-plus algebra restricted to natural numbers and $+\infty$. The resulting Kleene algebra is commutative. Similar variants can be obtained for max-plus algebras and other algebras ranging over the positive or negative integers.

instantiation *pnat* :: *commutative-kleene-algebra*
begin

definition *star-pnat* **where**
$x^\star \equiv (1::pnat)$

instance
proof
fix $x\ y\ z :: pnat$
show $1 + x \cdot x^\star \leq x^\star$
by (*metis star-pnat-def zero-pnat-top*)
show $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
by (*simp add*: *star-pnat-def*)
show $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
by (*simp add*: *star-pnat-def*)
show $x \cdot y = y \cdot x$
unfolding *times-pnat-def* **by** (*cases x*, *cases y*, *simp-all*)
qed

end

end

1.16 Domain Semirings

theory *Domain-Semiring*
imports *Kleene-Algebra*

begin

1.16.1 Domain Semigroups and Domain Monoids

class *domain-op* =
fixes *domain-op* :: $'a \Rightarrow 'a$ (d)

First we define the class of domain semigroups. Axioms are taken from [?].

class *domain-semigroup* = *semigroup-mult* + *domain-op* +
assumes *dsg1* [*simp*]: $d\ x \cdot x = x$
and *dsg2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
and *dsg3* [*simp*]: $d\ (d\ x \cdot y) = d\ x \cdot d\ y$
and *dsg4*: $d\ x \cdot d\ y = d\ y \cdot d\ x$

begin

lemma *domain-invol* [*simp*]: $d\ (d\ x) = d\ x$
proof −
  have $d\ (d\ x) = d\ (d\ (d\ x \cdot x))$
    by *simp*
  also have ... $= d\ (d\ x \cdot d\ x)$
    using *dsg3* **by** *presburger*
  also have ... $= d\ (d\ x \cdot x)$
    by *simp*
  finally show *?thesis*
    by *simp*
qed

The next lemmas show that domain elements form semilattices.

lemma *dom-el-idem* [*simp*]: $d\ x \cdot d\ x = d\ x$
proof −
  have $d\ x \cdot d\ x = d\ (d\ x \cdot x)$
    using *dsg3* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *dom-mult-closed* [*simp*]: $d\ (d\ x \cdot d\ y) = d\ x \cdot d\ y$
  by *simp*

lemma *dom-lc3* [*simp*]: $d\ x \cdot d\ (x \cdot y) = d\ (x \cdot y)$
proof −
  have $d\ x \cdot d\ (x \cdot y) = d\ (d\ x \cdot x \cdot y)$
    using *dsg3 mult-assoc* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *d-fixpoint*: $(\exists\, y.\ x = d\ y) \longleftrightarrow x = d\ x$
  by *auto*

lemma *d-type*: $\forall\, P.\ (\forall\, x.\ x = d\ x \longrightarrow P\ x) \longleftrightarrow (\forall\, x.\ P\ (d\ x))$
  by (*metis domain-invol*)

We define the semilattice ordering on domain semigroups and explore the semilattice of domain elements from the order point of view.

definition *ds-ord* :: $'a \Rightarrow\ 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*) **where**
  $x \sqsubseteq y \longleftrightarrow x = d\ x \cdot y$

lemma *ds-ord-refl*: $x \sqsubseteq x$
  by (*simp add*: *ds-ord-def*)

lemma *ds-ord-trans*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$
proof −
  assume $x \sqsubseteq y$ **and** a: $y \sqsubseteq z$
  hence b: $x = d\ x \cdot y$
    using *ds-ord-def* **by** *blast*
  hence $x = d\ x \cdot d\ y \cdot z$
    using *a ds-ord-def mult-assoc* **by** *force*
  also have ... $= d\ (d\ x \cdot y) \cdot z$
    by *simp*
  also have ... $= d\ x \cdot z$
    using *b* **by** *auto*
  finally show *?thesis*
    using *ds-ord-def* **by** *blast*
qed

lemma *ds-ord-antisym*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$
proof −
  assume a: $x \sqsubseteq y$ **and** $y \sqsubseteq x$
  hence b: $y = d\ y \cdot x$
    using *ds-ord-def* **by** *auto*
  have $x = d\ x \cdot d\ y \cdot x$
    using *a b ds-ord-def mult-assoc* **by** *force*
  also have ... $= d\ y \cdot x$
    by (*metis* (*full-types*) *b dsg3 dsg4*)
  thus *?thesis*
    using *b calculation* **by** *presburger*
qed

This relation is indeed an order.

sublocale *ds*: *order* $(\sqsubseteq)$ $\lambda x\ y.\ (x \sqsubseteq y \wedge x \neq y)$
proof
  show $\bigwedge x\ y.\ (x \sqsubseteq y \wedge x \neq y) = (x \sqsubseteq y \wedge \neg\ y \sqsubseteq x)$
    using *ds-ord-antisym* **by** *blast*
  show $\bigwedge x.\ x \sqsubseteq x$
    by (*rule ds-ord-refl*)
  show $\bigwedge x\ y\ z.\ x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$
    by (*rule ds-ord-trans*)
  show $\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$
    by (*rule ds-ord-antisym*)
qed

lemma *ds-ord-eq*: $x \sqsubseteq d\ x \longleftrightarrow x = d\ x$
  by (*simp add*: *ds-ord-def*)

lemma $x \sqsubseteq y \Longrightarrow z \cdot x \sqsubseteq z \cdot y$

oops

lemma *ds-ord-iso-right*: $x \sqsubseteq y \Longrightarrow x \cdot z \sqsubseteq y \cdot z$

proof –
  assume $x \sqsubseteq y$
  hence a: $x = d\ x \cdot y$
    by (*simp add*: *ds-ord-def*)
  hence $x \cdot z = d\ x \cdot y \cdot z$
    by *auto*
  also have ... $= d\ (d\ x \cdot y \cdot z) \cdot d\ x \cdot y \cdot z$
    using *dsg1 mult-assoc* **by** *presburger*
  also have ... $= d\ (x \cdot z) \cdot d\ x \cdot y \cdot z$
    using *a* **by** *presburger*
  finally show *?thesis*
    using *ds-ord-def dsg4 mult-assoc* **by** *auto*
qed

The order on domain elements could as well be defined based on multiplication/meet.

lemma *ds-ord-sl-ord*: $d\ x \sqsubseteq d\ y \longleftrightarrow d\ x \cdot d\ y = d\ x$
  using *ds-ord-def* **by** *auto*

lemma *ds-ord-1*: $d\ (x \cdot y) \sqsubseteq d\ x$
  by (*simp add*: *ds-ord-sl-ord dsg4*)

lemma *ds-subid-aux*: $d\ x \cdot y \sqsubseteq y$
  by (*simp add*: *ds-ord-def mult-assoc*)

lemma $y \cdot d\ x \sqsubseteq y$

oops

lemma *ds-dom-iso*: $x \sqsubseteq y \Longrightarrow d\ x \sqsubseteq d\ y$
proof –
  assume $x \sqsubseteq y$
  hence $x = d\ x \cdot y$
    by (*simp add*: *ds-ord-def*)
  hence $d\ x = d\ (d\ x \cdot y)$
    by *presburger*
  also have ... $= d\ x \cdot d\ y$
    by *simp*
  finally show *?thesis*
    using *ds-ord-sl-ord* **by** *auto*
qed

lemma *ds-dom-llp*: $x \sqsubseteq d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$
proof
  assume $x \sqsubseteq d\ y \cdot x$
  hence $x = d\ y \cdot x$
    by (*simp add*: *ds-subid-aux ds.order.antisym*)
  hence $d\ x = d\ (d\ y \cdot x)$
    by *presburger*

thus $d\ x \sqsubseteq d\ y$
using *ds-ord-sl-ord dsg4* **by** *force*
next
assume $d\ x \sqsubseteq d\ y$
thus $x \sqsubseteq d\ y \cdot x$
by (*metis* (*no-types*) *ds-ord-iso-right dsg1*)
qed

lemma *ds-dom-llp-strong*: $x = d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$
by (*simp add*: *ds-dom-llp ds.eq-iff ds-subid-aux*)

definition *refines* :: $'a \Rightarrow 'a \Rightarrow bool$
where *refines* $x\ y \equiv d\ y \sqsubseteq d\ x \wedge (d\ y) \cdot x \sqsubseteq y$

lemma *refines-refl*: *refines* $x\ x$
using *refines-def* **by** *simp*

lemma *refines-trans*: *refines* $x\ y \Longrightarrow$ *refines* $y\ z \Longrightarrow$ *refines* $x\ z$
unfolding *refines-def*
by (*metis domain-invol ds.dual-order.trans dsg1 dsg3 ds-ord-def*)

lemma *refines-antisym*: *refines* $x\ y \Longrightarrow$ *refines* $y\ x \Longrightarrow x = y$
unfolding *refines-def*
using *ds-dom-llp ds-ord-antisym* **by** *fastforce*

sublocale *ref*: *order refines* $\lambda x\ y.$ (*refines* $x\ y \wedge x \neq y$)
proof
show $\bigwedge x\ y.$ (*refines* $x\ y \wedge x \neq y$) $=$ (*refines* $x\ y \wedge \neg$ *refines* $y\ x$)
using *refines-antisym* **by** *blast*
show $\bigwedge x.$ *refines* $x\ x$
by (*rule refines-refl*)
show $\bigwedge x\ y\ z.$ *refines* $x\ y \Longrightarrow$ *refines* $y\ z \Longrightarrow$ *refines* $x\ z$
by (*rule refines-trans*)
show $\bigwedge x\ y.$ *refines* $x\ y \Longrightarrow$ *refines* $y\ x \Longrightarrow x = y$
by (*rule refines-antisym*)
qed

end

We expand domain semigroups to domain monoids.

class *domain-monoid* = *monoid-mult* + *domain-semigroup*
begin

lemma *dom-one* [*simp*]: $d\ 1 = 1$
proof −
have $1 = d\ 1 \cdot 1$
using *dsg1* **by** *presburger*
thus *?thesis*
by *simp*

qed

lemma *ds-subid-eq*: $x \sqsubseteq 1 \longleftrightarrow x = d\ x$
  by (*simp add*: *ds-ord-def*)

end

1.16.2 Domain Near-Semirings

The axioms for domain near-semirings are taken from [**?**].

class *domain-near-semiring* = *ab-near-semiring* + *plus-ord* + *domain-op* +
  assumes *dns1* [*simp*]: $d\ x \cdot x = x$
  and *dns2* [*simp*]: $d\ (x \cdot d\ y) = d(x \cdot y)$
  and *dns3* [*simp*]: $d\ (x + y) = d\ x + d\ y$
  and *dns4*: $d\ x \cdot d\ y = d\ y \cdot d\ x$
  and *dns5* [*simp*]: $d\ x \cdot (d\ x + d\ y) = d\ x$

begin

Domain near-semirings are automatically dioids; addition is idempotent.

subclass *near-dioid*
proof
  show $\bigwedge x.\ x + x = x$
  proof −
    fix x
    have *a*: $d\ x = d\ x \cdot d\ (x + x)$
      using *dns3 dns5* **by** *presburger*
    have $d\ (x + x) = d\ (x + x + (x + x)) \cdot d\ (x + x)$
      by (*metis* (*no-types*) *dns3 dns4 dns5*)
    hence $d\ (x + x) = d\ (x + x) + d\ (x + x)$
      by *simp*
    thus $x + x = x$
      by (*metis a dns1 dns4 distrib-right′*)
  qed
qed

Next we prepare to show that domain near-semirings are domain semigroups.

lemma *dom-iso*: $x \le y \Longrightarrow d\ x \le d\ y$
  using *order-prop* **by** *auto*

lemma *dom-add-closed* [*simp*]: $d\ (d\ x + d\ y) = d\ x + d\ y$
proof −
  have $d\ (d\ x + d\ y) = d\ (d\ x) + d\ (d\ y)$
    by *simp*
  thus *?thesis*
    by (*metis dns1 dns2 dns3 dns4*)
qed

lemma *dom-absorp-2* [*simp*]: $d\ x + d\ x \cdot d\ y = d\ x$
proof −
  have $d\ x + d\ x \cdot d\ y = d\ x \cdot d\ x + d\ x \cdot d\ y$
    by (*metis add-idem′ dns5*)
  also have ... $= (d\ x + d\ y) \cdot d\ x$
    by (*simp add*: *dns4*)
  also have ... $= d\ x \cdot (d\ x + d\ y)$
    by (*metis dom-add-closed dns4*)
  finally show *?thesis*
    by *simp*
qed

lemma *dom-1*: $d\ (x \cdot y) \leq d\ x$
proof −
  have $d\ (x \cdot y) = d\ (d\ x \cdot d\ (x \cdot y))$
    by (*metis dns1 dns2 mult-assoc*)
  also have ... $\leq d\ (d\ x) + d\ (d\ x \cdot d\ (x \cdot y))$
    by *simp*
  also have ... $= d\ (d\ x + d\ x \cdot d\ (x \cdot y))$
    using *dns3* **by** *presburger*
  also have ... $= d\ (d\ x)$
    by *simp*
  finally show *?thesis*
    by (*metis dom-add-closed add-idem′*)
qed

lemma *dom-subid-aux2*: $d\ x \cdot y \leq y$
proof −
  have $d\ x \cdot y \leq d\ (x + d\ y) \cdot y$
    by (*simp add*: *mult-isor*)
  also have ... $= (d\ x + d\ (d\ y)) \cdot d\ y \cdot y$
    using *dns1 dns3 mult-assoc* **by** *presburger*
  also have ... $= (d\ y + d\ y \cdot d\ x) \cdot y$
    by (*simp add*: *dns4 add-commute*)
  finally show *?thesis*
    by *simp*
qed

lemma *dom-glb*: $d\ x \leq d\ y \Longrightarrow d\ x \leq d\ z \Longrightarrow d\ x \leq d\ y \cdot d\ z$
  by (*metis dns5 less-eq-def mult-isor*)

lemma *dom-glb-eq*: $d\ x \leq d\ y \cdot d\ z \longleftrightarrow d\ x \leq d\ y \wedge d\ x \leq d\ z$
proof −
  have $d\ x \leq d\ z \longrightarrow d\ x \leq d\ z$
    by *meson*
  then show *?thesis*
   by (*metis* (*no-types*) *dom-absorp-2 dom-glb dom-subid-aux2 local.dual-order.trans local.join.sup.coboundedI2*)
qed

lemma *dom-ord*: $d\ x \leq d\ y \longleftrightarrow d\ x \cdot d\ y = d\ x$
proof
  assume $d\ x \leq d\ y$
  hence $d\ x + d\ y = d\ y$
    by (*simp add*: *less-eq-def*)
  thus $d\ x \cdot d\ y = d\ x$
    by (*metis dns5*)
next
  assume $d\ x \cdot d\ y = d\ x$
  thus $d\ x \leq d\ y$
    by (*metis dom-subid-aux2*)
qed

lemma *dom-export* [*simp*]: $d\ (d\ x \cdot y) = d\ x \cdot d\ y$
proof (*rule antisym*)
  have $d\ (d\ x \cdot y) = d\ (d\ (d\ x \cdot y)) \cdot d\ (d\ x \cdot y)$
    using *dns1* **by** *presburger*
  also have ... $= d\ (d\ x \cdot d\ y) \cdot d\ (d\ x \cdot y)$
    by (*metis dns1 dns2 mult-assoc*)
  finally show *a*: $d\ (d\ x \cdot y) \leq d\ x \cdot d\ y$
    by (*metis* (*no-types*) *dom-add-closed dom-glb dom-1 add-idem′ dns2 dns4*)
  have $d\ (d\ x \cdot y) = d\ (d\ x \cdot y) \cdot d\ x$
    using *a dom-glb-eq dom-ord* **by** *force*
  hence $d\ x \cdot d\ y = d\ (d\ x \cdot y) \cdot d\ y$
    by (*metis dns1 dns2 mult-assoc*)
  thus $d\ x \cdot d\ y \leq d\ (d\ x \cdot y)$
    using *a dom-glb-eq dom-ord* **by** *auto*
qed

subclass *domain-semigroup*
 by (*unfold-locales*, *auto simp*: *dns4*)

We compare the domain semigroup ordering with that of the dioid.

lemma *d-two-orders*: $d\ x \sqsubseteq d\ y \longleftrightarrow d\ x \leq d\ y$
  by (*simp add*: *dom-ord ds-ord-sl-ord*)

lemma *two-orders*: $x \sqsubseteq y \Longrightarrow x \leq y$
  by (*metis dom-subid-aux2 ds-ord-def*)

lemma $x \leq y \Longrightarrow x \sqsubseteq y$

oops

Next we prove additional properties.

lemma *dom-subdist*: $d\ x \leq d\ (x + y)$
  by *simp*

lemma *dom-distrib*: $d\ x + d\ y \cdot d\ z = (d\ x + d\ y) \cdot (d\ x + d\ z)$

proof −
  have $(d\ x + d\ y) \cdot (d\ x + d\ z) = d\ x \cdot (d\ x + d\ z) + d\ y \cdot (d\ x + d\ z)$
    using *distrib-right'* **by** *blast*
  also have ... $= d\ x + (d\ x + d\ z) \cdot d\ y$
    by (*metis* (*no-types*) *dns3 dns5 dsg4*)
  also have ... $= d\ x + d\ x \cdot d\ y + d\ z \cdot d\ y$
    using *add-assoc' distrib-right'* **by** *presburger*
  finally show *?thesis*
    by (*simp add*: *dsg4*)
qed

lemma *dom-llp1*: $x \le d\ y \cdot x \Longrightarrow d\ x \le d\ y$
proof −
  assume $x \le d\ y \cdot x$
  hence $d\ x \le d\ (d\ y \cdot x)$
    using *dom-iso* **by** *blast*
  also have ... $= d\ y \cdot d\ x$
    by *simp*
  finally show $d\ x \le d\ y$
    by (*simp add*: *dom-glb-eq*)
qed

lemma *dom-llp2*: $d\ x \le d\ y \Longrightarrow x \le d\ y \cdot x$
  using *d-two-orders local.ds-dom-llp two-orders* **by** *blast*

lemma *dom-llp*: $x \le d\ y \cdot x \longleftrightarrow d\ x \le d\ y$
  using *dom-llp1 dom-llp2* **by** *blast*

end

We expand domain near-semirings by an additive unit, using slightly different axioms.

class *domain-near-semiring-one* = *ab-near-semiring-one* + *plus-ord* + *domain-op*
+
  assumes *dnso1* [*simp*]: $x + d\ x \cdot x = d\ x \cdot x$
  and *dnso2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
  and *dnso3* [*simp*]: $d\ x + 1 = 1$
  and *dnso4* [*simp*]: $d\ (x + y) = d\ x + d\ y$
  and *dnso5*: $d\ x \cdot d\ y = d\ y \cdot d\ x$

begin

The previous axioms are derivable.

subclass *domain-near-semiring*
proof
  show *a*: $\bigwedge x.\ d\ x \cdot x = x$
   by (*metis add-commute local.dnso3 local.distrib-right' local.dnso1 local.mult-onel*)
  show $\bigwedge x\ y.\ d\ (x \cdot d\ y) = d\ (x \cdot y)$
    by *simp*

show $\bigwedge x\ y.\ d\ (x + y) = d\ x + d\ y$
  by *simp*
show $\bigwedge x\ y.\ d\ x \cdot d\ y = d\ y \cdot d\ x$
  by (*simp add*: *dnso5*)
show $\bigwedge x\ y.\ d\ x \cdot (d\ x + d\ y) = d\ x$
proof −
  fix *x y*
  have $\bigwedge x.\ 1 + d\ x = 1$
    using *add-commute dnso3* **by** *presburger*
  thus $d\ x \cdot (d\ x + d\ y) = d\ x$
    by (*metis* (*no-types*) *a dnso2 dnso4 dnso5 distrib-right′ mult-onel*)
qed
qed

subclass *domain-monoid* **..**

lemma *dom-subid*: $d\ x \leq 1$
  by (*simp add*: *less-eq-def*)

end

We add a left unit of multiplication.

class *domain-near-semiring-one-zerol* = *ab-near-semiring-one-zerol* + *domain-near-semiring-one* +
  assumes *dnso6* [*simp*]: $d\ 0 = 0$

begin

lemma *domain-very-strict*: $d\ x = 0 \longleftrightarrow x = 0$
  by (*metis annil dns1 dnso6*)

lemma *dom-weakly-local*: $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$
proof −
  have $x \cdot y = 0 \longleftrightarrow d\ (x \cdot y) = 0$
    by (*simp add*: *domain-very-strict*)
  also have ... $\longleftrightarrow d\ (x \cdot d\ y) = 0$
    by *simp*
  finally show *?thesis*
    using *domain-very-strict* **by** *blast*
qed

end

1.16.3 Domain Pre-Dioids

Pre-semirings with one and a left zero are automatically dioids. Hence there is no point defining domain pre-semirings separately from domain dioids. The axioms are once again from [**?**].

class *domain-pre-dioid-one* = *pre-dioid-one* + *domain-op* +

assumes *dpd1* : $x \leq d\ x \cdot x$
and *dpd2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
and *dpd3* [*simp*]: $d\ x \leq 1$
and *dpd4* [*simp*]: $d\ (x + y) = d\ x + d\ y$

begin

We prepare to show that every domain pre-dioid with one is a domain near-dioid with one.

lemma *dns1''* [*simp*]: $d\ x \cdot x = x$
proof (*rule antisym*)
show $d\ x \cdot x \leq x$
using *dpd3 mult-isor* **by** *fastforce*
show $x \leq d\ x \cdot x$
by (*simp add*: *dpd1*)
qed

lemma *d-iso*: $x \leq y \Longrightarrow d\ x \leq d\ y$
by (*metis dpd4 less-eq-def*)

lemma *domain-1''*: $d\ (x \cdot y) \leq d\ x$
proof –
have $d\ (x \cdot y) = d\ (x \cdot d\ y)$
by *simp*
also have ... $\leq d\ (x \cdot 1)$
by (*meson d-iso dpd3 mult-isol*)
finally show *?thesis*
by *simp*
qed

lemma *domain-export''* [*simp*]: $d\ (d\ x \cdot y) = d\ x \cdot d\ y$
proof (*rule antisym*)
have *one*: $d\ (d\ x \cdot y) \leq d\ x$
by (*metis dpd2 domain-1'' mult-onel*)
have *two*: $d\ (d\ x \cdot y) \leq d\ y$
using *d-iso dpd3 mult-isor* **by** *fastforce*
have $d\ (d\ x \cdot y) = d\ (d\ (d\ x \cdot y)) \cdot d\ (d\ x \cdot y)$
by *simp*
also have ... $= d\ (d\ x \cdot y) \cdot d\ (d\ x \cdot y)$
by (*metis dns1'' dpd2 mult-assoc*)
thus $d\ (d\ x \cdot y) \leq d\ x \cdot d\ y$
using *mult-isol-var one two* **by** *force*
next
have $d\ x \cdot d\ y \leq 1$
by (*metis dpd3 mult-1-right mult-isol order.trans*)
thus $d\ x \cdot d\ y \leq d\ (d\ x \cdot y)$
by (*metis dns1'' dpd2 mult-isol mult-oner*)
qed

lemma *dom-subid-aux1''*: $d\ x \cdot y \leq y$
proof −
  have $d\ x \cdot y \leq 1 \cdot y$
    using *dpd3 mult-isor* **by** *blast*
  thus *?thesis*
    by *simp*
qed

lemma *dom-subid-aux2''*: $x \cdot d\ y \leq x$
  using *dpd3 mult-isol* **by** *fastforce*

lemma *d-comm*: $d\ x \cdot d\ y = d\ y \cdot d\ x$
proof (*rule antisym*)
  have $d\ x \cdot d\ y = (d\ x \cdot d\ y) \cdot (d\ x \cdot d\ y)$
    by (*metis dns1'' domain-export''*)
  thus $d\ x \cdot d\ y \leq d\ y \cdot d\ x$
    by (*metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var*)
next
  have $d\ y \cdot d\ x = (d\ y \cdot d\ x) \cdot (d\ y \cdot d\ x)$
    by (*metis dns1'' domain-export''*)
  thus $d\ y \cdot d\ x \leq d\ x \cdot d\ y$
    by (*metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var*)
qed

subclass *domain-near-semiring-one*
  by (*unfold-locales*, *auto simp*: *d-comm local.join.sup.absorb2*)

lemma *domain-subid*: $x \leq 1 \Longrightarrow x \leq d\ x$
  by (*metis dns1 mult-isol mult-oner*)

lemma *d-preserves-equation*: $d\ y \cdot x \leq x \cdot d\ z \longleftrightarrow d\ y \cdot x = d\ y \cdot x \cdot d\ z$
  by (*metis dom-subid-aux2'' local.antisym local.dom-el-idem local.dom-subid-aux2 local.order-prop local.subdistl mult-assoc*)

lemma *d-restrict-iff*: $(x \leq y) \longleftrightarrow (x \leq d\ x \cdot y)$
  by (*metis dom-subid-aux2 dsg1 less-eq-def order-trans subdistl*)

lemma *d-restrict-iff-1*: $(d\ x \cdot y \leq z) \longleftrightarrow (d\ x \cdot y \leq d\ x \cdot z)$
  by (*metis dom-subid-aux2 domain-1'' domain-invol dsg1 mult-isol-var order-trans*)

end

We add once more a left unit of multiplication.

class *domain-pre-dioid-one-zerol* = *domain-pre-dioid-one* + *pre-dioid-one-zerol* +
  assumes *dpd5* [*simp*]: $d\ 0 = 0$

begin

subclass *domain-near-semiring-one-zerol*

by (*unfold-locales*, *simp*)

end

1.16.4 Domain Semirings

We do not consider domain semirings without units separately at the moment. The axioms are taken from from [**?**]

class *domain-semiringl* = *semiring-one-zerol* + *plus-ord* + *domain-op* +
assumes *dsr1* [*simp*]: $x + d\ x \cdot x = d\ x \cdot x$
and *dsr2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
and *dsr3* [*simp*]: $d\ x + 1 = 1$
and *dsr4* [*simp*]: $d\ 0 = 0$
and *dsr5* [*simp*]: $d\ (x + y) = d\ x + d\ y$

begin

Every domain semiring is automatically a domain pre-dioid with one and left zero.

subclass *dioid-one-zerol*
by (*standard*, *metis add-commute dsr1 dsr3 distrib-left mult-oner*)

subclass *domain-pre-dioid-one-zerol*
by (*standard*, *auto simp*: *less-eq-def*)

end

class *domain-semiring* = *domain-semiringl* + *semiring-one-zero*

1.16.5 The Algebra of Domain Elements

We show that the domain elements of a domain semiring form a distributive lattice. Unfortunately we cannot prove this within the type class of domain semirings.

typedef (**overloaded**) *'a d-element* = $\{x :: {}'a :: \textit{domain-semiring}.\ x = d\ x\}$
by (*rule-tac* $x = 1$ **in** *exI*, *simp add*: *domain-subid order-class.eq-iff*)

setup-lifting *type-definition-d-element*

instantiation *d-element* :: (*domain-semiring*) *bounded-lattice*

begin

lift-definition *less-eq-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *bool* **is** (≤) **.**

lift-definition *less-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *bool* **is** (<) **.**

lift-definition *bot-d-element* :: *'a d-element* **is** *0*
  by *simp*

lift-definition *top-d-element* :: *'a d-element* **is** *1*
  by *simp*

lift-definition *inf-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *'a d-element* **is** (·)
  by (*metis dsg3*)

lift-definition *sup-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *'a d-element* **is** (+)
  by *simp*

instance
  apply (*standard*; *transfer*)
  apply (*simp add*: *less-le-not-le*)+
  apply (*metis dom-subid-aux2''*)
  apply (*metis dom-subid-aux2*)
  apply (*metis dom-glb*)
  apply *simp*+
  by (*metis dom-subid*)

end

instance *d-element* :: (*domain-semiring*) *distrib-lattice*
  by (*standard*, *transfer*, *metis dom-distrib*)

1.16.6 Domain Semirings with a Greatest Element

If there is a greatest element in the semiring, then we have another equality.

class *domain-semiring-top* = *domain-semiring* + *order-top*

begin

notation *top* (⊤)

lemma *kat-equivalence-greatest*: $d\ x \leq d\ y \longleftrightarrow x \leq d\ y \cdot \top$
proof
  assume $d\ x \leq d\ y$
  thus $x \leq d\ y \cdot \top$
    by (*metis dsg1 mult-isol-var top-greatest*)
next
  assume $x \leq d\ y \cdot \top$
  thus $d\ x \leq d\ y$
    using *dom-glb-eq dom-iso* **by** *fastforce*
qed

end

1.16.7 Forward Diamond Operators

context *domain-semiringl*

begin

We define a forward diamond operator over a domain semiring. A more modular consideration is not given at the moment.

definition *fd* :: $'a \Rightarrow 'a \Rightarrow 'a$ ((|-⟩ -) [*61*,*81*] *82*) **where**
 $|x\rangle\ y = d\ (x \cdot y)$

lemma *fdia-d-simp* [*simp*]: $|x\rangle\ d\ y = |x\rangle\ y$
 by (*simp add*: *fd-def*)

lemma *fdia-dom* [*simp*]: $|x\rangle\ 1 = d\ x$
 by (*simp add*: *fd-def*)

lemma *fdia-add1*: $|x\rangle\ (y + z) = |x\rangle\ y + |x\rangle\ z$
 by (*simp add*: *fd-def distrib-left*)

lemma *fdia-add2*: $|x + y\rangle\ z = |x\rangle\ z + |y\rangle\ z$
 by (*simp add*: *fd-def distrib-right*)

lemma *fdia-mult*: $|x \cdot y\rangle\ z = |x\rangle\ |y\rangle\ z$
 by (*simp add*: *fd-def mult-assoc*)

lemma *fdia-one* [*simp*]: $|1\rangle\ x = d\ x$
 by (*simp add*: *fd-def*)

lemma *fdemodalisation1*: $d\ z \cdot |x\rangle\ y = 0 \longleftrightarrow d\ z \cdot x \cdot d\ y = 0$
proof –
 have $d\ z \cdot |x\rangle\ y = 0 \longleftrightarrow d\ z \cdot d\ (x \cdot y) = 0$
  by (*simp add*: *fd-def*)
 also have ... $\longleftrightarrow d\ z \cdot x \cdot y = 0$
  by (*metis annil dnso6 dsg1 dsg3 mult-assoc*)
 finally show *?thesis*
  using *dom-weakly-local* **by** *auto*
qed

lemma *fdemodalisation2*: $|x\rangle\ y \leq d\ z \longleftrightarrow x \cdot d\ y \leq d\ z \cdot x$
proof
 assume $|x\rangle\ y \leq d\ z$
 hence *a*: $d\ (x \cdot d\ y) \leq d\ z$
  by (*simp add*: *fd-def*)
 have $x \cdot d\ y = d\ (x \cdot d\ y) \cdot x \cdot d\ y$
  using *dsg1 mult-assoc* **by** *presburger*
 also have ... $\leq d\ z \cdot x \cdot d\ y$
  using *a calculation dom-llp2 mult-assoc* **by** *auto*
 finally show $x \cdot d\ y \leq d\ z \cdot x$
  using *dom-subid-aux2''* *order-trans* **by** *blast*

next
  assume $x \cdot d\ y \leq d\ z \cdot x$
  hence $d\ (x \cdot d\ y) \leq d\ (d\ z \cdot d\ x)$
    using *dom-iso* **by** *fastforce*
  also have ... $\leq d\ (d\ z)$
    using *domain-1''* **by** *blast*
  finally show $|x\rangle\ y \leq d\ z$
    by (*simp add*: *fd-def*)
qed

lemma *fd-iso1*: $d\ x \leq d\ y \Longrightarrow |z\rangle\ x \leq |z\rangle\ y$
  using *fd-def local.dom-iso local.mult-isol* **by** *fastforce*

lemma *fd-iso2*: $x \leq y \Longrightarrow |x\rangle\ z \leq |y\rangle\ z$
  by (*simp add*: *fd-def dom-iso mult-isor*)

lemma *fd-zero-var* [*simp*]: $|0\rangle\ x = 0$
  by (*simp add*: *fd-def*)

lemma *fd-subdist-1*: $|x\rangle\ y \leq |x\rangle\ (y + z)$
  by (*simp add*: *fd-iso1*)

lemma *fd-subdist-2*: $|x\rangle\ (d\ y \cdot d\ z) \leq |x\rangle\ y$
  by (*simp add*: *fd-iso1 dom-subid-aux2''*)

lemma *fd-subdist*: $|x\rangle\ (d\ y \cdot d\ z) \leq |x\rangle\ y \cdot |x\rangle\ z$
  using *fd-def fd-iso1 fd-subdist-2 dom-glb dom-subid-aux2* **by** *auto*

lemma *fdia-export-1*: $d\ y \cdot |x\rangle\ z = |d\ y \cdot x\rangle\ z$
  by (*simp add*: *fd-def mult-assoc*)

end

context *domain-semiring*

begin

lemma *fdia-zero* [*simp*]: $|x\rangle\ 0 = 0$
  by (*simp add*: *fd-def*)

end

1.16.8 Domain Kleene Algebras

We add the Kleene star to our considerations. Special domain axioms are not needed.

class *domain-left-kleene-algebra* = *left-kleene-algebra-zerol* + *domain-semiringl*

begin

lemma *dom-star* [*simp*]: $d\ (x^\star) = 1$
proof −
  have $d\ (x^\star) = d\ (1 + x \cdot x^\star)$
    by *simp*
  also have ... $= d\ 1 + d\ (x \cdot x^\star)$
    using *dns3* **by** *blast*
  finally show *?thesis*
    using *add-commute local.dsr3* **by** *auto*
qed

lemma *fdia-star-unfold* [*simp*]: $|1\rangle\ y + |x\rangle\ |x^\star\rangle\ y = |x^\star\rangle\ y$
proof −
  have $|1\rangle\ y + |x\rangle\ |x^\star\rangle\ y = |1 + x \cdot x^\star\rangle\ y$
    using *local.fdia-add2 local.fdia-mult* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *fdia-star-unfoldr* [*simp*]: $|1\rangle\ y + |x^\star\rangle\ |x\rangle\ y = |x^\star\rangle\ y$
proof −
  have $|1\rangle\ y + |x^\star\rangle\ |x\rangle\ y = |1 + x^\star \cdot x\rangle\ y$
    using *fdia-add2 fdia-mult* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *fdia-star-unfold-var* [*simp*]: $d\ y + |x\rangle\ |x^\star\rangle\ y = |x^\star\rangle\ y$
proof −
  have $d\ y + |x\rangle\ |x^\star\rangle\ y = |1\rangle\ y + |x\rangle\ |x^\star\rangle\ y$
    by *simp*
  also have ... $= |1 + x \cdot x^\star\rangle\ y$
    using *fdia-add2 fdia-mult* **by** *presburger*
  finally show *?thesis*
    by *simp*
qed

lemma *fdia-star-unfoldr-var* [*simp*]: $d\ y + |x^\star\rangle\ |x\rangle\ y = |x^\star\rangle\ y$
proof −
  have $d\ y + |x^\star\rangle\ |x\rangle\ y = |1\rangle\ y + |x^\star\rangle\ |x\rangle\ y$
    by *simp*
  also have ... $= |1 + x^\star \cdot x\rangle\ y$
    using *fdia-add2 fdia-mult* **by** *presburger*
  finally show *?thesis*
    by *simp*
qed

lemma *fdia-star-induct-var*: $|x\rangle\ y \le d\ y \Longrightarrow |x^\star\rangle\ y \le d\ y$
proof −

assume *a1*: $|x\rangle\ y \leq d\ y$
hence $x \cdot d\ y \leq d\ y \cdot x$
by (*simp add*: *fdemodalisation2*)
hence $x^\star \cdot d\ y \leq d\ y \cdot x^\star$
by (*simp add*: *star-sim1*)
thus *?thesis*
by (*simp add*: *fdemodalisation2*)
qed

lemma *fdia-star-induct*: $d\ z + |x\rangle\ y \leq d\ y \Longrightarrow |x^\star\rangle\ z \leq d\ y$
proof −
assume *a*: $d\ z + |x\rangle\ y \leq d\ y$
hence *b*: $d\ z \leq d\ y$ **and** *c*: $|x\rangle\ y \leq d\ y$
apply (*simp add*: *local.join.le-supE*)
using *a* **by** *auto*
hence *d*: $|x^\star\rangle\ z \leq |x^\star\rangle\ y$
using *fd-def fd-iso1* **by** *auto*
have $|x^\star\rangle\ y \leq d\ y$
using *c fdia-star-induct-var* **by** *blast*
thus *?thesis*
using *d* **by** *fastforce*
qed

lemma *fdia-star-induct-eq*: $d\ z + |x\rangle\ y = d\ y \Longrightarrow |x^\star\rangle\ z \leq d\ y$
by (*simp add*: *fdia-star-induct*)

end

class *domain-kleene-algebra* = *kleene-algebra* + *domain-semiring*

begin

subclass *domain-left-kleene-algebra* **..**

end

end

1.17 Antidomain Semirings

theory *Antidomain-Semiring*
imports *Domain-Semiring*
begin

1.17.1 Antidomain Monoids

We axiomatise antidomain monoids, using the axioms of [?].

class *antidomain-op* =

fixes *antidomain-op* :: $'a \Rightarrow 'a$ (*ad*)

class *antidomain-left-monoid* = *monoid-mult* + *antidomain-op* +
assumes *am1* [*simp*]: $ad\ x \cdot x = ad\ 1$
and *am2*: $ad\ x \cdot ad\ y = ad\ y \cdot ad\ x$
and *am3* [*simp*]: $ad\ (ad\ x) \cdot x = x$
and *am4* [*simp*]: $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$
and *am5* [*simp*]: $ad\ (x \cdot y) \cdot x \cdot ad\ y = ad\ (x \cdot y) \cdot x$

begin

no-notation *domain-op* (*d*)
no-notation *zero-class.zero* (*0*)

We define a zero element and operations of domain and addition.

definition *a-zero* :: $'a$ (*0*) **where**
$0 = ad\ 1$

definition *am-d* :: $'a \Rightarrow 'a$ (*d*) **where**
$d\ x = ad\ (ad\ x)$

definition *am-add-op* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\oplus$ *65*) **where**
$x \oplus y \equiv ad\ (ad\ x \cdot ad\ y)$

lemma *a-d-zero* [*simp*]: $ad\ x \cdot d\ x = 0$
by (*metis am1 am2 a-zero-def am-d-def*)

lemma *a-d-one* [*simp*]: $d\ x \oplus ad\ x = 1$
by (*metis am1 am3 mult-1-right am-d-def am-add-op-def*)

lemma *n-annil* [*simp*]: $0 \cdot x = 0$
proof −
have $0 \cdot x = d\ x \cdot ad\ x \cdot x$
by (*simp add*: *a-zero-def am-d-def*)
also have ... $= d\ x \cdot 0$
by (*metis am1 mult-assoc a-zero-def*)
thus *?thesis*
by (*metis am1 am2 am3 mult-assoc a-zero-def*)
qed

lemma *a-mult-idem* [*simp*]: $ad\ x \cdot ad\ x = ad\ x$
proof −
have $ad\ x \cdot ad\ x = ad\ (1 \cdot x) \cdot 1 \cdot ad\ x$
by *simp*
also have ... $= ad\ (1 \cdot x) \cdot 1$
using *am5* **by** *blast*
finally show *?thesis*
by *simp*
qed

lemma *a-add-idem* [*simp*]: $ad\ x \oplus ad\ x = ad\ x$
by (*metis am1 am3 am4 mult-1-right am-add-op-def*)

The next three axioms suffice to show that the domain elements form a Boolean algebra.

lemma *a-add-comm*: $x \oplus y = y \oplus x$
using *am2 am-add-op-def* **by** *auto*

lemma *a-add-assoc*: $x \oplus (y \oplus z) = (x \oplus y) \oplus z$
proof –
have $\bigwedge x\ y.\ ad\ x \cdot ad\ (x \cdot y) = ad\ x$
by (*metis a-mult-idem am2 am4 mult-assoc*)
thus *?thesis*
by (*metis a-add-comm am-add-op-def local.am3 local.am4 mult-assoc*)
qed

lemma *huntington* [*simp*]: $ad\ (x \oplus y) \oplus ad\ (x \oplus ad\ y) = ad\ x$
using *a-add-idem am-add-op-def* **by** *auto*

lemma *a-absorb1* [*simp*]: $(ad\ x \oplus ad\ y) \cdot ad\ x = ad\ x$
by (*metis a-add-idem a-mult-idem am4 mult-assoc am-add-op-def*)

lemma *a-absorb2* [*simp*]: $ad\ x \oplus ad\ x \cdot ad\ y = ad\ x$
proof –
have $ad\ (ad\ x) \cdot ad\ (ad\ x \cdot ad\ y) = ad\ (ad\ x)$
by (*metis* (*no-types*) *a-mult-idem local.am4 local.mult.semigroup-axioms semigroup.assoc*)
then show *?thesis*
using *a-add-idem am-add-op-def* **by** *auto*
qed

The distributivity laws remain to be proved; our proofs follow those of Maddux [**?**].

lemma *prod-split* [*simp*]: $ad\ x \cdot ad\ y \oplus ad\ x \cdot d\ y = ad\ x$
using *a-add-idem am-d-def am-add-op-def* **by** *auto*

lemma *sum-split* [*simp*]: $(ad\ x \oplus ad\ y) \cdot (ad\ x \oplus d\ y) = ad\ x$
using *a-add-idem am-d-def am-add-op-def* **by** *fastforce*

lemma *a-comp-simp* [*simp*]: $(ad\ x \oplus ad\ y) \cdot d\ x = ad\ y \cdot d\ x$
proof –
have *f1*: $(ad\ x \oplus ad\ y) \cdot d\ x = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ (ad\ x) \cdot ad\ (ad\ (ad\ y))$
by (*simp add*: *am-add-op-def am-d-def*)
have *f2*: $ad\ y = ad\ (ad\ (ad\ y))$
using *a-add-idem am-add-op-def* **by** *auto*
have $ad\ y = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ y$
by (*metis* (*no-types*) *a-absorb1 a-add-comm am-add-op-def*)

then show *?thesis*
using *f2 f1* **by** (*simp add*: *am-d-def local.am2 local.mult.semigroup-axioms semigroup.assoc*)
qed

lemma *a-distrib1*: $ad\ x \cdot (ad\ y \oplus ad\ z) = ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ z$
proof −
have *f1*: $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a)) \cdot ad\ (ad\ a)) = ad\ a$
using *a-add-idem am-add-op-def* **by** *auto*
have *f2*: $\bigwedge a\ aa.\ ad\ ((a::'a) \cdot aa) \cdot (a \cdot ad\ aa) = ad\ (a \cdot aa) \cdot a$
using *local.am5 mult-assoc* **by** *auto*
have *f3*: $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using *f1* **by** *simp*
have $\bigwedge a.\ ad\ (a::'a) \cdot ad\ a = ad\ a$
by *simp*
then have $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a::'a) \cdot ad\ aa)) = ad\ aa \cdot ad\ a$
using *f3 f2* **by** (*metis* (*no-types*) *local.am2 local.am4 mult-assoc*)
then have $ad\ x \cdot (ad\ y \oplus ad\ z) = ad\ x \cdot (ad\ y \oplus ad\ z) \cdot ad\ y \oplus ad\ x \cdot (ad\ y \oplus ad\ z) \cdot d\ y$
using *am-add-op-def am-d-def local.am2 local.am4* **by** *presburger*
also have ... $= ad\ x \cdot ad\ y \oplus ad\ x \cdot (ad\ y \oplus ad\ z) \cdot d\ y$
by (*simp add*: *mult-assoc*)
also have ... $= ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ z \cdot d\ y$
by (*simp add*: *mult-assoc*)
also have ... $= ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ y \cdot ad\ z \oplus ad\ x \cdot ad\ z \cdot d\ y$
by (*metis a-add-idem a-mult-idem local.am4 mult-assoc am-add-op-def*)
also have ... $= ad\ x \cdot ad\ y \oplus (ad\ x \cdot ad\ z \cdot ad\ y \oplus ad\ x \cdot ad\ z \cdot d\ y)$
by (*metis am2 mult-assoc a-add-assoc*)
finally show *?thesis*
by (*metis a-add-idem a-mult-idem am4 am-d-def am-add-op-def*)
qed

lemma *a-distrib2*: $ad\ x \oplus ad\ y \cdot ad\ z = (ad\ x \oplus ad\ y) \cdot (ad\ x \oplus ad\ z)$
proof −
have *f1*: $\bigwedge a\ aa\ ab.\ ad\ (ad\ (ad\ (a::'a) \cdot ad\ aa) \cdot ad\ (ad\ a \cdot ad\ ab)) = ad\ a \cdot ad\ (ad\ (ad\ aa) \cdot ad\ (ad\ ab))$
using *a-distrib1 am-add-op-def* **by** *auto*
have $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
by (*metis a-absorb2 a-mult-idem am-add-op-def*)
then have $ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ (ad\ (ad\ x) \cdot ad\ (ad\ z)) = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y \cdot ad\ z))$
using *f1* **by** (*metis* (*full-types*))
then show *?thesis*
by (*simp add*: *am-add-op-def*)
qed

lemma *aa-loc* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
proof −
have *f1*: $x \cdot d\ y \cdot y = x \cdot y$

by (*metis am3 mult-assoc am-d-def*)
have *f2*: $\bigwedge w\ z.\ ad\ (w \cdot z) \cdot (w \cdot ad\ z) = ad\ (w \cdot z) \cdot w$
by (*metis am5 mult-assoc*)
hence *f3*: $\bigwedge z.\ ad\ (x \cdot y) \cdot (x \cdot z) = ad\ (x \cdot y) \cdot (x \cdot (ad\ (ad\ (ad\ y) \cdot y) \cdot z))$
using *f1* **by** (*metis* (*no-types*) *mult-assoc am-d-def*)
have $ad\ (x \cdot ad\ (ad\ y)) \cdot (x \cdot y) = 0$ **using** *f1*
by (*metis am1 mult-assoc n-annil a-zero-def am-d-def*)
thus *?thesis*
by (*metis a-d-zero am-d-def f3 local.am1 local.am2 local.am3 local.am4*)
qed

lemma *a-loc* [*simp*]: $ad\ (x \cdot d\ y) = ad\ (x \cdot y)$
proof −
have $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using *am-add-op-def am-d-def prod-split* **by** *auto*
then show *?thesis*
by (*metis* (*full-types*) *aa-loc am-d-def*)
qed

lemma *d-a-export* [*simp*]: $d\ (ad\ x \cdot y) = ad\ x \cdot d\ y$
proof −
have *f1*: $\bigwedge a\ aa.\ ad\ ((a::'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$
using *a-loc am-d-def* **by** *auto*
have $\bigwedge a.\ ad\ (ad\ (a::'a) \cdot a) = 1$
using *a-d-one am-add-op-def am-d-def* **by** *auto*
then have $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a::'a) \cdot ad\ aa)) = ad\ a \cdot ad\ aa$
using *f1* **by** (*metis a-distrib2 am-add-op-def local.mult-1-left*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *am-d-def*)
qed

Every antidomain monoid is a domain monoid.

sublocale *dm*: *domain-monoid am-d* $(\cdot)$ *1*
apply (*unfold-locales*)
apply (*simp add*: *am-d-def*)
apply *simp*
using *am-d-def d-a-export* **apply** *auto*[*1*]
by (*simp add*: *am-d-def local.am2*)

lemma *ds-ord-iso1*: $x \sqsubseteq y \Longrightarrow z \cdot x \sqsubseteq z \cdot y$

oops

lemma *a-very-costrict*: $ad\ x = 1 \longleftrightarrow x = 0$
proof
assume *a*: $ad\ x = 1$
hence $0 = ad\ x \cdot x$
using *a-zero-def* **by** *force*
thus $x = 0$

by (*simp add*: *a*)
next
assume $x = 0$
thus $ad\ x = 1$
using *a-zero-def am-d-def dm.dom-one* **by** *auto*
qed

lemma *a-weak-loc*: $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$
proof −
have $x \cdot y = 0 \longleftrightarrow ad\ (x \cdot y) = 1$
by (*simp add*: *a-very-costrict*)
also have ... $\longleftrightarrow ad\ (x \cdot d\ y) = 1$
by *simp*
finally show *?thesis*
using *a-very-costrict* **by** *blast*
qed

lemma *a-closure* [*simp*]: $d\ (ad\ x) = ad\ x$
using *a-add-idem am-add-op-def am-d-def* **by** *auto*

lemma *a-d-mult-closure* [*simp*]: $d\ (ad\ x \cdot ad\ y) = ad\ x \cdot ad\ y$
by *simp*

lemma *kat-3'*: $d\ x \cdot y \cdot ad\ z = 0 \Longrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
by (*metis dm.dom-one local.am5 local.mult-1-left a-zero-def am-d-def*)

lemma *s4* [*simp*]: $ad\ x \cdot ad\ (ad\ x \cdot y) = ad\ x \cdot ad\ y$
proof −
have $\bigwedge a\ aa.\ ad\ (a::'a) \cdot ad\ (ad\ aa) = ad\ (ad\ (ad\ a \cdot aa))$
using *am-d-def d-a-export* **by** *presburger*
then have $\bigwedge a\ aa.\ ad\ (ad\ (a::'a)) \cdot ad\ aa = ad\ (ad\ (ad\ aa \cdot a))$
using *local.am2* **by** *presburger*
then show *?thesis*
by (*metis a-comp-simp a-d-mult-closure am-add-op-def am-d-def local.am2*)
qed

end

class *antidomain-monoid* = *antidomain-left-monoid* +
assumes *am6* [*simp*]: $x \cdot ad\ 1 = ad\ 1$

begin

lemma *kat-3-equiv*: $d\ x \cdot y \cdot ad\ z = 0 \longleftrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
apply *standard*
apply (*metis kat-3'*)
by (*simp add*: *mult-assoc a-zero-def am-d-def*)

no-notation *a-zero* (*0*)

no-notation *am-d* (*d*)

end

1.17.2 Antidomain Near-Semirings

We define antidomain near-semirings. We do not consider units separately. The axioms are taken from [**?**].

notation *zero-class.zero* (*0*)

class *antidomain-near-semiring* = *ab-near-semiring-one-zerol* + *antidomain-op* + *plus-ord* +
  assumes *ans1* [*simp*]: $ad\ x \cdot x = 0$
  and *ans2* [*simp*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$
  and *ans3* [*simp*]: $ad\ (ad\ x) + ad\ x = 1$
  and *ans4* [*simp*]: $ad\ (x + y) = ad\ x \cdot ad\ y$

begin

definition *ans-d* :: $'a \Rightarrow 'a$ (*d*) **where**
   $d\ x = ad\ (ad\ x)$

lemma *a-a-one* [*simp*]: $d\ 1 = 1$
proof −
  have $d\ 1 = d\ 1 + 0$
    by *simp*
  also have ... $= d\ 1 + ad\ 1$
    by (*metis ans1 mult-1-right*)
  finally show *?thesis*
    by (*simp add*: *ans-d-def*)
qed

lemma *a-very-costrict'*: $ad\ x = 1 \longleftrightarrow x = 0$
proof
  assume $ad\ x = 1$
  hence $x = ad\ x \cdot x$
    by *simp*
  thus $x = 0$
    by *auto*
next
  assume $x = 0$
  hence $ad\ x = ad\ 0$
    by *blast*
  thus $ad\ x = 1$
    by (*metis a-a-one ans-d-def local.ans1 local.mult-1-right*)
qed

lemma *one-idem* [*simp*]: $1 + 1 = 1$
proof −

```
  have 1 + 1 = d 1 + d 1
    by simp
  also have ... = ad (ad 1 · 1) + ad (ad 1 · d 1)
    using a-a-one ans-d-def by auto
  also have ... = ad (ad 1 · d 1)
    using ans-d-def local.ans2 by presburger
  also have ... = ad (ad 1 · 1)
    by simp
  also have ... = d 1
    by (simp add: ans-d-def)
  finally show ?thesis
    by simp
qed
```

Every antidomain near-semiring is automatically a dioid, and therefore ordered.

```
subclass near-dioid-one-zerol
proof
  show ⋀x. x + x = x
  proof −
    fix x
    have x + x = 1 · x + 1 · x
      by simp
    also have ... = (1 + 1) · x
      using distrib-right′ by presburger
    finally show x + x = x
      by simp
  qed
qed

lemma d1-a [simp]: d x · x = x
proof −
  have x = (d x + ad x) · x
    by (simp add: ans-d-def)
  also have ... = d x · x + ad x · x
    using distrib-right′ by blast
  also have ... = d x · x + 0
    by simp
  finally show ?thesis
    by auto
qed

lemma a-comm: ad x · ad y = ad y · ad x
  using add-commute ans4 by fastforce

lemma a-subid: ad x ≤ 1
  using local.ans3 local.join.sup-ge2 by fastforce

lemma a-subid-aux1: ad x · y ≤ y
```

using *a-subid mult-isor* **by** *fastforce*

lemma *a-subdist*: $ad\ (x + y) \leq ad\ x$
by (*metis a-subid-aux1 ans4 add-comm*)

lemma *a-antitone*: $x \leq y \Longrightarrow ad\ y \leq ad\ x$
using *a-subdist local.order-prop* **by** *auto*

lemma *a-mul-d* [*simp*]: $ad\ x \cdot d\ x = 0$
by (*metis a-comm ans-d-def local.ans1*)

lemma *a-gla1*: $ad\ x \cdot y = 0 \Longrightarrow ad\ x \leq ad\ y$
proof −
assume $ad\ x \cdot y = 0$
hence *a*: $ad\ x \cdot d\ y = 0$
by (*metis a-subid a-very-costrict′ ans-d-def local.ans2 local.join.sup.order-iff*)
have $ad\ x = (d\ y + ad\ y\ ) \cdot ad\ x$
by (*simp add*: *ans-d-def*)
also have ... $= d\ y \cdot ad\ x + ad\ y \cdot ad\ x$
using *distrib-right′* **by** *blast*
also have ... $= ad\ x \cdot d\ y + ad\ x \cdot ad\ y$
using *a-comm ans-d-def* **by** *auto*
also have ... $= ad\ x \cdot ad\ y$
by (*simp add*: *a*)
finally show $ad\ x \leq ad\ y$
by (*metis a-subid-aux1*)
qed

lemma *a-gla2*: $ad\ x \leq ad\ y \Longrightarrow ad\ x \cdot y = 0$
proof −
assume $ad\ x \leq ad\ y$
hence $ad\ x \cdot y \leq ad\ y \cdot y$
using *mult-isor* **by** *blast*
thus *?thesis*
by (*simp add*: *join.le-bot*)
qed

lemma *a2-eq* [*simp*]: $ad\ (x \cdot d\ y) = ad\ (x \cdot y)$
proof (*rule antisym*)
show $ad\ (x \cdot y) \leq ad\ (x \cdot d\ y)$
by (*simp add*: *ans-d-def local.less-eq-def*)
next
show $ad\ (x \cdot d\ y) \leq ad\ (x \cdot y)$
by (*metis a-gla1 a-mul-d ans1 d1-a mult-assoc*)
qed

lemma *a-export′* [*simp*]: $ad\ (ad\ x \cdot y) = d\ x + ad\ y$
proof (*rule antisym*)
have $ad\ (ad\ x \cdot y) \cdot ad\ x \cdot d\ y = 0$

```
    by (simp add: a-gla2 local.mult.semigroup-axioms semigroup.assoc)
  hence a: ad (ad x · y) · d y ≤ ad (ad x)
    by (metis a-comm a-gla1 ans4 mult-assoc ans-d-def)
  have ad (ad x · y) = ad (ad x · y) · d y + ad (ad x · y) · ad y
   by (metis (no-types) add-commute ans3 ans4 distrib-right' mult-onel ans-d-def)
  thus ad (ad x · y) ≤ d x + ad y
    by (metis a-subid-aux1 a join.sup-mono ans-d-def)
next
  show d x + ad y ≤ ad (ad x · y)
    by (metis a2-eq a-antitone a-comm a-subid-aux1 join.sup-least ans-d-def)
qed
```

Every antidomain near-semiring is a domain near-semiring.

```
sublocale dnsz: domain-near-semiring-one-zerol (+) (·) 1 0 ans-d (≤) (<)
  apply (unfold-locales)
  apply simp
  using a2-eq ans-d-def apply auto[1]
  apply (simp add: a-subid ans-d-def local.join.sup-absorb2)
  apply (simp add: ans-d-def)
  apply (simp add: a-comm ans-d-def)
  using a-a-one a-very-costrict' ans-d-def by force

lemma a-idem [simp]: ad x · ad x = ad x
proof −
  have ad x = (d x + ad x ) · ad x
    by (simp add: ans-d-def)
  also have ... = d x · ad x + ad x · ad x
    using distrib-right' by blast
  finally show ?thesis
    by (simp add: ans-d-def)
qed

lemma a-3-var [simp]: ad x · ad y · (x + y) = 0
  by (metis ans1 ans4)

lemma a-3 [simp]: ad x · ad y · d (x + y) = 0
  by (metis a-mul-d ans4)

lemma a-closure' [simp]: d (ad x) = ad x
proof −
  have d (ad x) = ad (1 · d x)
    by (simp add: ans-d-def)
  also have ... = ad (1 · x)
    using a2-eq by blast
  finally show ?thesis
    by simp
qed
```

The following counterexamples show that some of the antidomain monoid axioms do not need to hold.

lemma $x \cdot ad\ 1 = ad\ 1$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

lemma *phl-seq-inv*: $d\ v \cdot x \cdot y \cdot ad\ w = 0 \Longrightarrow \exists z.\ d\ v \cdot x \cdot d\ z = 0 \wedge ad\ z \cdot y \cdot ad\ w = 0$
proof −
  assume $d\ v \cdot x \cdot y \cdot ad\ w = 0$
  hence $d\ v \cdot x \cdot d\ (y \cdot ad\ w) = 0 \wedge ad\ (y \cdot ad\ w) \cdot y \cdot ad\ w = 0$
    by (*metis dnsz.dom-weakly-local local.ans1 mult-assoc*)
  thus $\exists z.\ d\ v \cdot x \cdot d\ z = 0 \wedge ad\ z \cdot y \cdot ad\ w = 0$
    by *blast*
qed

lemma *a-fixpoint*: $ad\ x = x \Longrightarrow (\forall y.\ y = 0)$
proof −
  assume *a1*: $ad\ x = x$
  { fix $aa :: 'a$
    have $aa = 0$
      using *a1* **by** (*metis* (*no-types*) *a-mul-d ans-d-def local.annil local.ans3 local.join.sup.idem local.mult-1-left*)
  }
  then show *?thesis*
    by *blast*
qed

no-notation *ans-d* (*d*)

end

1.17.3 Antidomain Pre-Dioids

Antidomain pre-diods are based on a different set of axioms, which are again taken from [**?**].

class *antidomain-pre-dioid* = *pre-dioid-one-zerol* + *antidomain-op* +
  assumes *apd1* [*simp*]: $ad\ x \cdot x = 0$
  and *apd2* [*simp*]: $ad\ (x \cdot y) \leq ad\ (x \cdot ad\ (ad\ y))$
  and *apd3* [*simp*]: $ad\ (ad\ x) + ad\ x = 1$

begin

definition *apd-d* :: $'a \Rightarrow 'a$ (d) **where**
$d\ x = ad\ (ad\ x)$

lemma *a-very-costrict''*: $ad\ x = 1 \longleftrightarrow x = 0$
by (*metis add-commute local.add-zerol local.antisym local.apd1 local.apd3 local.join.bot-least local.mult-1-right local.phl-skip*)

lemma *a-subid'*: $ad\ x \leq 1$
using *local.apd3 local.join.sup-ge2* **by** *fastforce*

lemma *d1-a'* [*simp*]: $d\ x \cdot x = x$
proof −
have $x = (d\ x + ad\ x) \cdot x$
by (*simp add*: *apd-d-def*)
also have ... $= d\ x \cdot x + ad\ x \cdot x$
using *distrib-right'* **by** *blast*
also have ... $= d\ x \cdot x + 0$
by *simp*
finally show *?thesis*
by *auto*
qed

lemma *a-subid-aux1'*: $ad\ x \cdot y \leq y$
using *a-subid' mult-isor* **by** *fastforce*

lemma *a-mul-d'* [*simp*]: $ad\ x \cdot d\ x = 0$
proof −
have $1 = ad\ (ad\ x \cdot x)$
using *a-very-costrict''* **by** *force*
thus *?thesis*
by (*metis a-subid' a-very-costrict'' apd-d-def local.antisym local.apd2*)
qed

lemma *a-d-closed* [*simp*]: $d\ (ad\ x) = ad\ x$
proof (*rule antisym*)
have $d\ (ad\ x) = (d\ x + ad\ x) \cdot d\ (ad\ x)$
by (*simp add*: *apd-d-def*)
also have ... $= ad\ (ad\ x) \cdot ad\ (d\ x) + ad\ x \cdot d\ (ad\ x)$
using *apd-d-def local.distrib-right'* **by** *presburger*
also have ... $= ad\ x \cdot d\ (ad\ x)$
using *a-mul-d' apd-d-def* **by** *auto*
finally show $d\ (ad\ x) \leq ad\ x$
by (*metis a-subid' mult-1-right mult-isol apd-d-def*)
next
have $ad\ x = ad\ (1 \cdot x)$
by *simp*
also have ... $\leq ad\ (1 \cdot d\ x)$

```
    using apd-d-def local.apd2 by presburger
  also have ... = ad (d x)
    by simp
  finally show ad x ≤ d (ad x)
    by (simp add: apd-d-def)
qed

lemma meet-ord-def: ad x ≤ ad y ⟷ ad x · ad y = ad x
  by (metis a-d-closed a-subid-aux1' d1-a' eq-iff mult-1-right mult-isol)

lemma d-weak-loc: x · y = 0 ⟷ x · d y = 0
proof -
  have x · y = 0 ⟷ ad (x · y) = 1
    by (simp add: a-very-costrict'')
  also have ... ⟷ ad (x · d y) = 1
    by (metis apd1 apd2 a-subid' apd-d-def d1-a' eq-iff mult-1-left mult-assoc)
  finally show ?thesis
    by (simp add: a-very-costrict'')
qed

lemma gla-1: ad x · y = 0 ⟹ ad x ≤ ad y
proof -
  assume ad x · y = 0
  hence a: ad x · d y = 0
    using d-weak-loc by force
  hence d y = ad x · d y + d y
    by simp
  also have ... = (1 + ad x) · d y
    using join.sup-commute by auto
  also have ... = (d x + ad x) · d y
    using apd-d-def calculation by auto
  also have ... = d x · d y
    by (simp add: a join.sup-commute)
  finally have d y ≤ d x
    by (metis apd-d-def a-subid' mult-1-right mult-isol)
  hence d y · ad x = 0
    by (metis apd-d-def a-d-closed a-mul-d' distrib-right' less-eq-def no-trivial-inverse)
  hence ad x = ad y · ad x
    by (metis apd-d-def apd3 add-0-left distrib-right' mult-1-left)
  thus ad x ≤ ad y
    by (metis add-commute apd3 mult-oner subdistl)
qed

lemma a2-eq' [simp]: ad (x · d y) = ad (x · y)
proof (rule antisym)
  show ad (x · y) ≤ ad (x · d y)
    by (simp add: apd-d-def)
next
  show ad (x · d y) ≤ ad (x · y)
```

by (*metis gla-1 apd1 a-mul-d′ d1-a′ mult-assoc*)
qed

lemma *a-supdist-var*: $ad\ (x + y) \leq ad\ x$
by (*metis gla-1 apd1 join.le-bot subdistl*)

lemma *a-antitone′*: $x \leq y \Longrightarrow ad\ y \leq ad\ x$
using *a-supdist-var local.order-prop* **by** *auto*

lemma *a-comm-var*: $ad\ x \cdot ad\ y \leq ad\ y \cdot ad\ x$
proof −
have $ad\ x \cdot ad\ y = d\ (ad\ x \cdot ad\ y) \cdot ad\ x \cdot ad\ y$
by (*simp add*: *mult-assoc*)
also have ... $\leq d\ (ad\ x \cdot ad\ y) \cdot ad\ x$
using *a-subid′ mult-isol* **by** *fastforce*
also have ... $\leq d\ (ad\ y) \cdot ad\ x$
by (*simp add*: *a-antitone′ a-subid-aux1′ apd-d-def local.mult-isor*)
finally show *?thesis*
by *simp*
qed

lemma *a-comm′*: $ad\ x \cdot ad\ y = ad\ y \cdot ad\ x$
by (*simp add*: *a-comm-var eq-iff*)

lemma *a-closed* [*simp*]: $d\ (ad\ x \cdot ad\ y) = ad\ x \cdot ad\ y$
proof −
have *f1*: $\bigwedge x\ y.\ ad\ x \leq ad\ (ad\ y \cdot x)$
by (*simp add*: *a-antitone′ a-subid-aux1′*)
have $\bigwedge x\ y.\ d\ (ad\ x \cdot y) \leq ad\ x$
by (*metis a2-eq′ a-antitone′ a-comm′ a-d-closed apd-d-def f1*)
hence $\bigwedge x\ y.\ d\ (ad\ x \cdot y) \cdot y = ad\ x \cdot y$
by (*metis d1-a′ meet-ord-def mult-assoc apd-d-def*)
thus *?thesis*
by (*metis f1 a-comm′ apd-d-def meet-ord-def*)
qed

lemma *a-export″* [*simp*]: $ad\ (ad\ x \cdot y) = d\ x + ad\ y$
proof (*rule antisym*)
have $ad\ (ad\ x \cdot y) \cdot ad\ x \cdot d\ y = 0$
using *d-weak-loc mult-assoc* **by** *fastforce*
hence *a*: $ad\ (ad\ x \cdot y) \cdot d\ y \leq d\ x$
by (*metis a-closed a-comm′ apd-d-def gla-1 mult-assoc*)
have $ad\ (ad\ x \cdot y) = ad\ (ad\ x \cdot y) \cdot d\ y + ad\ (ad\ x \cdot y) \cdot ad\ y$
by (*metis apd3 a-comm′ d1-a′ distrib-right′ mult-1-right apd-d-def*)
thus $ad\ (ad\ x \cdot y) \leq d\ x + ad\ y$
by (*metis a-subid-aux1′ a join.sup-mono*)
next
have $ad\ y \leq ad\ (ad\ x \cdot y)$
by (*simp add*: *a-antitone′ a-subid-aux1′*)

thus $d\ x + ad\ y \leq ad\ (ad\ x \cdot y)$
  by (*metis apd-d-def a-mul-d′ d1-a′ gla-1 apd1 join.sup-least mult-assoc*)
qed

lemma *d1-sum-var*: $x + y \leq (d\ x + d\ y) \cdot (x + y)$
proof −
  have $x + y = d\ x \cdot x + d\ y \cdot y$
    by *simp*
  also have ... $\leq (d\ x + d\ y) \cdot x + (d\ x + d\ y) \cdot y$
    using *local.distrib-right′ local.join.sup-ge1 local.join.sup-ge2 local.join.sup-mono*
by *presburger*
  finally show *?thesis*
    using *order-trans subdistl-var* **by** *blast*
qed

lemma *a4′*: $ad\ (x + y) = ad\ x \cdot ad\ y$
proof (*rule antisym*)
  show $ad\ (x + y) \leq ad\ x \cdot ad\ y$
    by (*metis a-d-closed a-supdist-var add-commute d1-a′ local.mult-isol-var*)
  hence $ad\ x \cdot ad\ y = ad\ x \cdot ad\ y + ad\ (x + y)$
    using *less-eq-def add-commute* **by** *simp*
  also have ... $= ad\ (ad\ (ad\ x \cdot ad\ y) \cdot (x + y))$
    by (*metis a-closed a-export″*)
  finally show $ad\ x \cdot ad\ y \leq ad\ (x + y)$
    using *a-antitone′ apd-d-def d1-sum-var* **by** *auto*
qed

Antidomain pre-dioids are domain pre-dioids and antidomain near-semirings, but still not antidomain monoids.

sublocale *dpdz*: *domain-pre-dioid-one-zerol* $(+)\ (\cdot)\ 1\ 0\ (\leq)\ (<)\ \lambda x.\ ad\ (ad\ x)$
  apply (*unfold-locales*)
  using *apd-d-def d1-a′* **apply** *auto*[*1*]
  using *a2-eq′ apd-d-def* **apply** *auto*[*1*]
  apply (*simp add*: *a-subid′*)
  apply (*simp add*: *a4′ apd-d-def*)
  by (*metis a-mul-d′ a-very-costrict″ apd-d-def local.mult-onel*)

subclass *antidomain-near-semiring*
  apply (*unfold-locales*)
  apply *simp*
  using *local.apd2 local.less-eq-def* **apply** *blast*
  apply *simp*
  by (*simp add*: *a4′*)

lemma *a-supdist*: $ad\ (x + y) \leq ad\ x + ad\ y$
  using *a-supdist-var local.join.le-supI1* **by** *auto*

lemma *a-gla*: $ad\ x \cdot y = 0 \longleftrightarrow ad\ x \leq ad\ y$
  using *gla-1 a-gla2* **by** *blast*

lemma *a-subid-aux2*: $x \cdot ad\ y \leq x$
  using *a-subid′ mult-isol* **by** *fastforce*

lemma *a42-var*: $d\ x \cdot d\ y \leq ad\ (ad\ x + ad\ y)$
  by (*simp add*: *apd-d-def*)

lemma *d1-weak* [*simp*]: $(d\ x + d\ y) \cdot x = x$
proof −
  have $(d\ x + d\ y) \cdot x = (1 + d\ y) \cdot x$
    by *simp*
  thus *?thesis*
   by (*metis add-commute apd-d-def dpdz.dnso3 local.mult-1-left*)
qed

lemma $x \cdot ad\ 1 = ad\ 1$

oops

lemma $ad\ x \cdot (y + z) = ad\ x \cdot y + ad\ x \cdot z$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

no-notation *apd-d* (*d*)

end

1.17.4 Antidomain Semirings

Antidomain semirings are direct expansions of antidomain pre-dioids, but do not require idempotency of addition. Hence we give a slightly different axiomatisation, following [**?**].

class *antidomain-semiringl* = *semiring-one-zerol* + *plus-ord* + *antidomain-op* +
  assumes *as1* [*simp*]: $ad\ x \cdot x = 0$
  and *as2* [*simp*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$
  and *as3* [*simp*]: $ad\ (ad\ x) + ad\ x = 1$

begin

definition *ads-d* :: $'a \Rightarrow 'a$ (*d*) **where**
  $d\ x = ad\ (ad\ x)$

lemma *one-idem′*: *1* + *1* = *1*
by (*metis as1 as2 as3 add-zeror mult.right-neutral*)

Every antidomain semiring is a dioid and an antidomain pre-dioid.

subclass *dioid*
by (*standard*, *metis distrib-left mult.right-neutral one-idem′*)

subclass *antidomain-pre-dioid*
by (*unfold-locales*, *auto simp*: *local.less-eq-def*)

lemma *am5-lem* [*simp*]: *ad* (*x* · *y*) · *ad* (*x* · *ad y*) = *ad x*
proof −
have *ad* (*x* · *y*) · *ad* (*x* · *ad y*) = *ad* (*x* · *d y*) · *ad* (*x* · *ad y*)
using *ads-d-def local.a2-eq′ local.apd-d-def* **by** *auto*
also have ... = *ad* (*x* · *d y* + *x* · *ad y*)
using *ans4* **by** *presburger*
also have ... = *ad* (*x* · (*d y* + *ad y*))
using *distrib-left* **by** *presburger*
finally show *?thesis*
by (*simp add*: *ads-d-def*)
qed

lemma *am6-lem* [*simp*]: *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x*
proof −
fix *x y*
have *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x* · *ad y* + *0*
by *simp*
also have ... = *ad* (*x* · *y*) · *x* · *ad y* + *ad* (*x* · *d y*) · *x* · *d y*
using *ans1 mult-assoc* **by** *presburger*
also have ... = *ad* (*x* · *y*) · *x* · (*ad y* + *d y*)
using *ads-d-def local.a2-eq′ local.apd-d-def local.distrib-left* **by** *auto*
finally show *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x*
using *add-commute ads-d-def local.as3* **by** *auto*
qed

lemma *a-zero* [*simp*]: *ad 0* = *1*
by (*simp add*: *local.a-very-costrict′′*)

lemma *a-one* [*simp*]: *ad 1* = *0*
using *a-zero local.dpdz.dpd5* **by** *blast*

subclass *antidomain-left-monoid*
by (*unfold-locales*, *auto simp*: *local.a-comm′*)

Every antidomain left semiring is a domain left semiring.

no-notation *domain-semiringl-class.fd* ((|-⟩ -) [*61*,*81*] *82*)

definition *fdia* :: ′*a* ⇒ ′*a* ⇒ ′*a* ((|-⟩ -) [*61*,*81*] *82*) **where**

$|x\rangle\ y = ad\ (ad\ (x \cdot y))$

sublocale *ds*: *domain-semiringl* $(+)$ $(\cdot)$ *1 0* $\lambda x.\ ad\ (ad\ x)$ $(\leq)$ $(<)$
rewrites *ds.fd x y* $\equiv$ *fdia x y*
proof –
show *class.domain-semiringl* $(+)$ $(\cdot)$ *1 0* $(\lambda x.\ ad\ (ad\ x))$ $(\leq)$ $(<)$
by (*unfold-locales*, *auto simp*: *local.dpdz.dpd4 ans-d-def*)
then interpret *ds*: *domain-semiringl* $(+)$ $(\cdot)$ *1 0* $\lambda x.\ ad\ (ad\ x)$ $(\leq)$ $(<)$ **.**
show *ds.fd x y* $\equiv$ *fdia x y*
by (*auto simp*: *fdia-def ds.fd-def*)
qed

lemma *fd-eq-fdia* [*simp*]: *domain-semiringl.fd* $(\cdot)$ *d x y* $\equiv$ *fdia x y*
proof –
have *class.domain-semiringl* $(+)$ $(\cdot)$ *1 0 d* $(\leq)$ $(<)$
by (*unfold-locales*, *auto simp*: *ads-d-def local.ans-d-def*)
hence *domain-semiringl.fd* $(\cdot)$ *d x y* $= d\ ((\cdot)\ x\ y)$
by (*rule domain-semiringl.fd-def*)
also have ... $=$ *ds.fd x y*
by (*simp add*: *ds.fd-def ads-d-def*)
finally show *domain-semiringl.fd* $(\cdot)$ *d x y* $\equiv |x\rangle\ y$
by *auto*
qed

end

class *antidomain-semiring* $=$ *antidomain-semiringl* $+$ *semiring-one-zero*

begin

Every antidomain semiring is an antidomain monoid.

subclass *antidomain-monoid*
by (*standard*, *metis ans1 mult-1-right annir*)

lemma *a-zero* $= 0$
by (*simp add*: *local.a-zero-def*)

sublocale *ds*: *domain-semiring* $(+)$ $(\cdot)$ *1 0* $\lambda x.\ ad\ (ad\ x)$ $(\leq)$ $(<)$
rewrites *ds.fd x y* $\equiv$ *fdia x y*
by *unfold-locales*

end

1.17.5 The Boolean Algebra of Domain Elements

typedef (**overloaded**) *'a a2-element* $= \{x :: 'a :: antidomain\text{-}semiring.\ x = d\ x\}$
by (*rule-tac x=1* **in** *exI*, *auto simp*: *ads-d-def*)

setup-lifting *type-definition-a2-element*

instantiation *a2-element* :: (*antidomain-semiring*) *boolean-algebra*

begin

lift-definition *less-eq-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* $\Rightarrow$ *bool* **is** $(\leq)$
.

lift-definition *less-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* $\Rightarrow$ *bool* **is** $(<)$ **.**

lift-definition *bot-a2-element* :: $'a$ *a2-element* **is** *0*
  by (*simp add*: *ads-d-def*)

lift-definition *top-a2-element* :: $'a$ *a2-element* **is** *1*
  by (*simp add*: *ads-d-def*)

lift-definition *inf-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element*
is $(\cdot)$
  by (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-mult-closed*)

lift-definition *sup-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element*
is $(+)$
  by (*metis ads-d-def ds.dsr5*)

lift-definition *minus-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element*
is $\lambda x\ y.\ x \cdot ad\ y$
  by (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.domain-export''*)

lift-definition *uminus-a2-element* :: $'a$ *a2-element* $\Rightarrow$ $'a$ *a2-element* **is** *antidomain-op*
  by (*simp add*: *ads-d-def*)

instance
  apply (*standard*; *transfer*)
  apply (*simp add*: *less-le-not-le*)
  apply *simp*
  apply *auto*[*1*]
  apply *simp*
  apply (*metis a-subid-aux2 ads-d-def*)
  apply (*metis a-subid-aux1' ads-d-def*)
  apply (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-glb*)
  apply *simp*
  apply *simp*
  apply *simp*
  apply *simp*
  apply (*metis a-subid' ads-d-def*)
  apply (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-distrib*)
  apply (*metis ads-d-def ans1*)
  apply (*metis ads-d-def ans3*)
  by *simp*

end

1.17.6 Further Properties

context *antidomain-semiringl*

begin

lemma *a-2-var*: $ad\ x \cdot d\ y = 0 \longleftrightarrow ad\ x \leq ad\ y$
  using *local.a-gla local.ads-d-def local.dpdz.dom-weakly-local* **by** *auto*

The following two lemmas give the Galois connection of Heyting algebras.

lemma *da-shunt1*: $x \leq d\ y + z \Longrightarrow x \cdot ad\ y \leq z$
proof −
  assume $x \leq d\ y + z$
  hence $x \cdot ad\ y \leq (d\ y + z) \cdot ad\ y$
    using *mult-isor* **by** *blast*
  also have ... $= d\ y \cdot ad\ y + z \cdot ad\ y$
    by *simp*
  also have ... $\leq z$
    by (*simp add*: *a-subid-aux2 ads-d-def*)
  finally show $x \cdot ad\ y \leq z$
    by *simp*
qed

lemma *da-shunt2*: $x \leq ad\ y + z \Longrightarrow x \cdot d\ y \leq z$
  using *da-shunt1 local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *d-a-galois1*: $d\ x \cdot ad\ y \leq d\ z \longleftrightarrow d\ x \leq d\ z + d\ y$
  by (*metis add-assoc local.a-gla local.ads-d-def local.am2 local.ans4 local.ans-d-def local.dnsz.dnso4*)

lemma *d-a-galois2*: $d\ x \cdot d\ y \leq d\ z \longleftrightarrow d\ x \leq d\ z + ad\ y$
proof −
  have $\bigwedge a\ aa.\ ad\ ((a{::}'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$
    using *local.a2-eq′ local.apd-d-def* **by** *force*
  then show *?thesis*
    by (*metis d-a-galois1 local.a-export′ local.ads-d-def local.ans-d-def*)
qed

lemma *d-cancellation-1*: $d\ x \leq d\ y + d\ x \cdot ad\ y$
proof −
  have *a*: $d\ (d\ x \cdot ad\ y) = ad\ y \cdot d\ x$
    using *local.a-closure′ local.ads-d-def local.am2 local.ans-d-def* **by** *auto*
  hence $d\ x \leq d\ (d\ x \cdot ad\ y) + d\ y$
    using *d-a-galois1 local.a-comm-var local.ads-d-def* **by** *fastforce*
  thus *?thesis*
    using *a add-commute local.ads-d-def local.am2* **by** *auto*

qed

lemma *d-cancellation-2*: $(d\ z + d\ y) \cdot ad\ y \leq d\ z$
  by (*simp add*: *da-shunt1*)

lemma *a-de-morgan*: $ad\ (ad\ x \cdot ad\ y) = d\ (x + y)$
  by (*simp add*: *local.ads-d-def*)

lemma *a-de-morgan-var-3*: $ad\ (d\ x + d\ y) = ad\ x \cdot ad\ y$
  using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *a-de-morgan-var-4*: $ad\ (d\ x \cdot d\ y) = ad\ x + ad\ y$
  using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *a-4*: $ad\ x \leq ad\ (x \cdot y)$
  using *local.a-add-idem local.a-antitone′ local.dpdz.domain-1′′ am-add-op-def* **by** *fastforce*

lemma *a-6*: $ad\ (d\ x \cdot y) = ad\ x + ad\ y$
  using *a-de-morgan-var-4 local.ads-d-def* **by** *auto*

lemma *a-7*: $d\ x \cdot ad\ (d\ y + d\ z) = d\ x \cdot ad\ y \cdot ad\ z$
  using *a-de-morgan-var-3 local.mult.semigroup-axioms semigroup.assoc* **by** *fastforce*

lemma *a-d-add-closure* [*simp*]: $d\ (ad\ x + ad\ y) = ad\ x + ad\ y$
  using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *d-6* [*simp*]: $d\ x + ad\ x \cdot d\ y = d\ x + d\ y$
proof −
  have $ad\ (ad\ x \cdot (x + ad\ y)) = d\ (x + y)$
    by (*simp add*: *distrib-left ads-d-def*)
  thus *?thesis*
    by (*simp add*: *local.ads-d-def local.ans-d-def*)
qed

lemma *d-7* [*simp*]: $ad\ x + d\ x \cdot ad\ y = ad\ x + ad\ y$
  by (*metis a-d-add-closure local.ads-d-def local.ans4 local.s4*)

lemma *a-mult-add*: $ad\ x \cdot (y + x) = ad\ x \cdot y$
  by (*simp add*: *distrib-left*)

lemma *kat-2*: $y \cdot ad\ z \leq ad\ x \cdot y \Longrightarrow d\ x \cdot y \cdot ad\ z = 0$
proof −
  assume *a*: $y \cdot ad\ z \leq ad\ x \cdot y$
  hence $d\ x \cdot y \cdot ad\ z \leq d\ x \cdot ad\ x \cdot y$
    using *local.mult-isol mult-assoc* **by** *presburger*
  thus *?thesis*
    using *local.join.le-bot ads-d-def* **by** *auto*
qed

lemma *kat-3*: $d\ x \cdot y \cdot ad\ z = 0 \Longrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
  using *local.a-zero-def local.ads-d-def local.am-d-def local.kat-3'* **by** *auto*

lemma *kat-4*: $d\ x \cdot y = d\ x \cdot y \cdot d\ z \Longrightarrow d\ x \cdot y \leq y \cdot d\ z$
  using *a-subid-aux1 mult-assoc ads-d-def* **by** *auto*

lemma *kat-2-equiv*: $y \cdot ad\ z \leq ad\ x \cdot y \longleftrightarrow d\ x \cdot y \cdot ad\ z = 0$
proof
  assume $y \cdot ad\ z \leq ad\ x \cdot y$
  thus $d\ x \cdot y \cdot ad\ z = 0$
    by (*simp add*: *kat-2*)
next
  assume *1*: $d\ x \cdot y \cdot ad\ z = 0$
  have $y \cdot ad\ z = (d\ x + ad\ x) \cdot y \cdot ad\ z$
    by (*simp add*: *local.ads-d-def*)
  also have ... $= d\ x \cdot y \cdot ad\ z + ad\ x \cdot y \cdot ad\ z$
    using *local.distrib-right* **by** *presburger*
  also have ... $= ad\ x \cdot y \cdot ad\ z$
    using *1* **by** *auto*
  also have ... $\leq ad\ x \cdot y$
    by (*simp add*: *local.a-subid-aux2*)
  finally show $y \cdot ad\ z \leq ad\ x \cdot y$ **.**
qed

lemma *kat-4-equiv*: $d\ x \cdot y = d\ x \cdot y \cdot d\ z \longleftrightarrow d\ x \cdot y \leq y \cdot d\ z$
  using *local.ads-d-def local.dpdz.d-preserves-equation* **by** *auto*

lemma *kat-3-equiv-opp*: $ad\ z \cdot y \cdot d\ x = 0 \longleftrightarrow y \cdot d\ x = d\ z \cdot y \cdot d\ x$
proof –
  have $ad\ z \cdot (y \cdot d\ x) = 0 \longrightarrow (ad\ z \cdot y \cdot d\ x = 0) = (y \cdot d\ x = d\ z \cdot y \cdot d\ x)$
    by (*metis* (*no-types*, *hide-lams*) *add-commute local.add-zerol local.ads-d-def local.as3 local.distrib-right' local.mult-1-left mult-assoc*)
  thus *?thesis*
   by (*metis a-4 local.a-add-idem local.a-gla2 local.ads-d-def mult-assoc am-add-op-def*)
qed

lemma *kat-4-equiv-opp*: $y \cdot d\ x = d\ z \cdot y \cdot d\ x \longleftrightarrow y \cdot d\ x \leq d\ z \cdot y$
  using *kat-2-equiv kat-3-equiv-opp local.ads-d-def* **by** *auto*

1.17.7 Forward Box and Diamond Operators

lemma *fdemodalisation22*: $|x\rangle\ y \leq d\ z \longleftrightarrow ad\ z \cdot x \cdot d\ y = 0$
proof –
  have $|x\rangle\ y \leq d\ z \longleftrightarrow d\ (x \cdot y) \leq d\ z$
    by (*simp add*: *fdia-def ads-d-def*)
  also have ... $\longleftrightarrow ad\ z \cdot d\ (x \cdot y) = 0$
    by (*metis add-commute local.a-gla local.ads-d-def local.ans4*)
  also have ... $\longleftrightarrow ad\ z \cdot x \cdot y = 0$

using *dpdz.dom-weakly-local mult-assoc ads-d-def* **by** *auto*
finally show *?thesis*
using *dpdz.dom-weakly-local ads-d-def* **by** *auto*
qed

lemma *dia-diff-var*: $|x\rangle\ y \leq |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ z$
proof −
have *1*: $|x\rangle\ (d\ y \cdot d\ z) \leq |x\rangle\ (1 \cdot d\ z)$
using *dpdz.dom-glb-eq ds.fd-subdist fdia-def ads-d-def* **by** *force*
have $|x\rangle\ y = |x\rangle\ (d\ y \cdot (ad\ z + d\ z))$
by (*metis as3 add-comm ds.fdia-d-simp mult-1-right ads-d-def*)
also have ... $= |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ (d\ y \cdot d\ z)$
by (*simp add*: *local.distrib-left local.ds.fdia-add1*)
also have ... $\leq |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ (1 \cdot d\ z)$
using *1 local.join.sup.mono* **by** *blast*
finally show *?thesis*
by (*simp add*: *fdia-def ads-d-def*)
qed

lemma *dia-diff*: $|x\rangle\ y \cdot ad\ (\ |x\rangle\ z\ ) \leq |x\rangle\ (d\ y \cdot ad\ z)$
using *fdia-def dia-diff-var d-a-galois2 ads-d-def* **by** *metis*

lemma *fdia-export-2*: $ad\ y \cdot |x\rangle\ z = |ad\ y \cdot x\rangle\ z$
using *local.am-d-def local.d-a-export local.fdia-def mult-assoc* **by** *auto*

lemma *fdia-split*: $|x\rangle\ y = d\ z \cdot |x\rangle\ y + ad\ z \cdot |x\rangle\ y$
by (*metis mult-onel ans3 distrib-right ads-d-def*)

definition *fbox* :: $'a \Rightarrow 'a \Rightarrow 'a$ ((|-] -) [*61*,*81*] *82*) **where**
$|x]\ y = ad\ (x \cdot ad\ y)$

The next lemmas establish the De Morgan duality between boxes and diamonds.

lemma *fdia-fbox-de-morgan-2*: $ad\ (\ |x\rangle\ y) = |x]\ ad\ y$
using *fbox-def local.a-closure local.a-loc local.am-d-def local.fdia-def* **by** *auto*

lemma *fbox-simp*: $|x]\ y = |x]\ d\ y$
using *fbox-def local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *fbox-dom* [*simp*]: $|x]\ 0 = ad\ x$
by (*simp add*: *fbox-def*)

lemma *fbox-add1*: $|x]\ (d\ y \cdot d\ z) = |x]\ y \cdot |x]\ z$
using *a-de-morgan-var-4 fbox-def local.distrib-left* **by** *auto*

lemma *fbox-add2*: $|x + y]\ z = |x]\ z \cdot |y]\ z$
by (*simp add*: *fbox-def*)

lemma *fbox-mult*: $|x \cdot y]\ z = |x]\ |y]\ z$

using *fbox-def local.a2-eq′ local.apd-d-def mult-assoc* **by** *auto*

lemma *fbox-zero* [*simp*]: $|0]\ x = 1$
by (*simp add*: *fbox-def*)

lemma *fbox-one* [*simp*]: $|1]\ x = d\ x$
by (*simp add*: *fbox-def ads-d-def*)

lemma *fbox-iso*: $d\ x \leq d\ y \Longrightarrow |z]\ x \leq |z]\ y$
proof −
assume $d\ x \leq d\ y$
hence $ad\ y \leq ad\ x$
using *local.a-add-idem local.a-antitone′ local.ads-d-def am-add-op-def* **by** *fastforce*
hence $z \cdot ad\ y \leq z \cdot ad\ x$
by (*simp add*: *mult-isol*)
thus $|z]\ x \leq |z]\ y$
by (*simp add*: *fbox-def a-antitone′*)
qed

lemma *fbox-antitone-var*: $x \leq y \Longrightarrow |y]\ z \leq |x]\ z$
by (*simp add*: *fbox-def a-antitone mult-isor*)

lemma *fbox-subdist-1*: $|x]\ (d\ y \cdot d\ z) \leq |x]\ y$
using *a-de-morgan-var-4 fbox-def local.a-supdist-var local.distrib-left* **by** *force*

lemma *fbox-subdist-2*: $|x]\ y \leq |x]\ (d\ y + d\ z)$
by (*simp add*: *fbox-iso ads-d-def*)

lemma *fbox-subdist*: $|x]\ d\ y + |x]\ d\ z \leq |x]\ (d\ y + d\ z)$
by (*simp add*: *fbox-iso sup-least ads-d-def*)

lemma *fbox-diff-var*: $|x]\ (d\ y + ad\ z) \cdot |x]\ z \leq |x]\ y$
proof −
have $ad\ (ad\ y) \cdot ad\ (ad\ z) = ad\ (ad\ z + ad\ y)$
using *local.dpdz.dsg4* **by** *auto*
then have $d\ (d\ (d\ y + ad\ z) \cdot d\ z) \leq d\ y$
by (*simp add*: *local.a-subid-aux1′ local.ads-d-def*)
then show *?thesis*
by (*metis fbox-add1 fbox-iso*)
qed

lemma *fbox-diff*: $|x]\ (d\ y + ad\ z) \leq |x]\ y + ad\ (\ |x]\ z\ )$
proof −
have *f1*: $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using *local.a-closure′ local.ans-d-def* **by** *force*
have *f2*: $\bigwedge a\ aa.\ ad\ (ad\ (a::'a)) + ad\ aa = ad\ (ad\ a \cdot aa)$
using *local.ans-d-def* **by** *auto*
have *f3*: $\bigwedge a\ aa.\ ad\ ((a::'a) + aa) = ad\ (aa + a)$
by (*simp add*: *local.am2*)

then have *f4*: $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a{::}'a) \cdot aa)) = ad\ (ad\ aa + a)$
  using *f2 f1* **by** (*metis* (*no-types*) *local.ans4*)
have *f5*: $\bigwedge a\ aa\ ab.\ ad\ ((a{::}'a) \cdot (aa + ab)) = ad\ (a \cdot (ab + aa))$
  using *f3 local.distrib-left* **by** *presburger*
have *f6*: $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a{::}'a) + aa)) = ad\ (ad\ aa \cdot a)$
  using *f3 f1* **by** *fastforce*
have $ad\ (x \cdot ad\ (y + ad\ z)) \leq ad\ (ad\ (x \cdot ad\ z) \cdot (x \cdot ad\ y))$
  using *f5 f2 f1* **by** (*metis* (*no-types*) *a-mult-add fbox-def fbox-subdist-1 local.a-gla2 local.ads-d-def local.ans4 local.distrib-left local.gla-1 mult-assoc*)
then show *?thesis*
  using *f6 f4 f3 f1* **by** (*simp add*: *fbox-def local.ads-d-def*)
qed

end

context *antidomain-semiring*

begin

lemma *kat-1*: $d\ x \cdot y \leq y \cdot d\ z \Longrightarrow y \cdot ad\ z \leq ad\ x \cdot y$
proof −
assume *a*: $d\ x \cdot y \leq y \cdot d\ z$
have $y \cdot ad\ z = d\ x \cdot y \cdot ad\ z + ad\ x \cdot y \cdot ad\ z$
  by (*metis local.ads-d-def local.as3 local.distrib-right local.mult-1-left*)
also have ... $\leq y \cdot (d\ z \cdot ad\ z) + ad\ x \cdot y \cdot ad\ z$
  by (*metis a add-iso mult-isor mult-assoc*)
also have ... $= ad\ x \cdot y \cdot ad\ z$
  by (*simp add*: *ads-d-def*)
finally show $y \cdot ad\ z \leq ad\ x \cdot y$
  using *local.a-subid-aux2 local.dual-order.trans* **by** *blast*
qed

lemma *kat-1-equiv*: $d\ x \cdot y \leq y \cdot d\ z \longleftrightarrow y \cdot ad\ z \leq ad\ x \cdot y$
  using *kat-1 kat-2 kat-3 kat-4* **by** *blast*

lemma *kat-3-equiv'*: $d\ x \cdot y \cdot ad\ z = 0 \longleftrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
  by (*simp add*: *kat-1-equiv local.kat-2-equiv local.kat-4-equiv*)

lemma *kat-1-equiv-opp*: $y \cdot d\ x \leq d\ z \cdot y \longleftrightarrow ad\ z \cdot y \leq y \cdot ad\ x$
  by (*metis kat-1-equiv local.a-closure' local.ads-d-def local.ans-d-def*)

lemma *kat-2-equiv-opp*: $ad\ z \cdot y \leq y \cdot ad\ x \longleftrightarrow ad\ z \cdot y \cdot d\ x = 0$
  by (*simp add*: *kat-1-equiv-opp local.kat-3-equiv-opp local.kat-4-equiv-opp*)

lemma *fbox-one-1* [*simp*]: $|x]\ 1 = 1$
  by (*simp add*: *fbox-def*)

lemma *fbox-demodalisation3*: $d\ y \leq |x]\ d\ z \longleftrightarrow d\ y \cdot x \leq x \cdot d\ z$
  by (*simp add*: *fbox-def a-gla kat-2-equiv-opp mult-assoc ads-d-def*)

end

1.17.8 Antidomain Kleene Algebras

class *antidomain-left-kleene-algebra* = *antidomain-semiringl* + *left-kleene-algebra-zerol*

begin

sublocale *dka*: *domain-left-kleene-algebra* (+) (·) *1 0 d* (≤) (<) *star*
rewrites *domain-semiringl.fd* (·) *d x y* ≡ $|x\rangle$ *y*
by (*unfold-locales*, *auto simp add*: *local.ads-d-def ans-d-def*)

lemma *a-star* [*simp*]: *ad* ($x^\star$) = *0*
using *dka.dom-star local.a-very-costrict''* *local.ads-d-def* **by** *force*

lemma *fbox-star-unfold* [*simp*]: $|1]$ *z* · $|x]$ $|x^\star]$ *z* = $|x^\star]$ *z*
proof −
have *ad* (*ad z* + *x* · ($x^\star$ · *ad z*)) = *ad* ($x^\star$ · *ad z*)
using *local.conway.dagger-unfoldl-distr mult-assoc* **by** *auto*
then show *?thesis*
using *local.a-closure'* *local.ans-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2*
by *fastforce*
qed

lemma *fbox-star-unfold-var* [*simp*]: *d z* · $|x]$ $|x^\star]$ *z* = $|x^\star]$ *z*
using *fbox-star-unfold* **by** *auto*

lemma *fbox-star-unfoldr* [*simp*]: $|1]$ *z* · $|x^\star]$ $|x]$ *z* = $|x^\star]$ *z*
by (*metis fbox-star-unfold fbox-mult star-slide-var*)

lemma *fbox-star-unfoldr-var* [*simp*]: *d z* · $|x^\star]$ $|x]$ *z* = $|x^\star]$ *z*
using *fbox-star-unfoldr* **by** *auto*

lemma *fbox-star-induct-var*: *d y* ≤ $|x]$ *y* ⟹ *d y* ≤ $|x^\star]$ *y*
proof −
assume *a1*: *d y* ≤ $|x]$ *y*
have ⋀*a*. *ad* (*ad* (*ad* (*a*::*'a*))) = *ad a*
using *local.a-closure'* *local.ans-d-def* **by** *auto*
then have *ad* (*ad* ($x^\star$ · *ad y*)) ≤ *ad y*
using *a1* **by** (*metis dka.fdia-star-induct local.a-export'* *local.ads-d-def local.ans4 local.ans-d-def local.eq-refl local.fbox-def local.fdia-def local.meet-ord-def*)
then have *ad* (*ad y* + *ad* ($x^\star$ · *ad y*)) = *zero-class.zero*
by (*metis* (*no-types*) *add-commute local.a-2-var local.ads-d-def local.ans4*)
then show *?thesis*
using *local.a-2-var local.ads-d-def local.fbox-def* **by** *auto*
qed

lemma *fbox-star-induct*: *d y* ≤ *d z* · $|x]$ *y* ⟹ *d y* ≤ $|x^\star]$ *z*

proof −
  assume *a1*: $d\ y \leq d\ z \cdot |x]\ y$
  hence *a*: $d\ y \leq d\ z$ **and** $d\ y \leq |x]\ y$
    apply (*metis local.a-subid-aux2 local.dual-order.trans local.fbox-def*)
    using *a1 dka.dom-subid-aux2 local.dual-order.trans* **by** *blast*
  hence $d\ y \leq |x^\star]\ y$
    using *fbox-star-induct-var* **by** *blast*
  thus *?thesis*
    using *a local.fbox-iso local.order.trans* **by** *blast*
qed

lemma *fbox-star-induct-eq*: $d\ z \cdot |x]\ y = d\ y \Longrightarrow d\ y \leq |x^\star]\ z$
  by (*simp add*: *fbox-star-induct*)

lemma *fbox-export-1*: $ad\ y + |x]\ y = |d\ y \cdot x]\ y$
  by (*simp add*: *local.a-6 local.fbox-def mult-assoc*)

lemma *fbox-export-2*: $d\ y + |x]\ y = |ad\ y \cdot x]\ y$
  by (*simp add*: *local.ads-d-def local.ans-d-def local.fbox-def mult-assoc*)

end

class *antidomain-kleene-algebra* = *antidomain-semiring* + *kleene-algebra*

begin

subclass *antidomain-left-kleene-algebra* **..**

lemma $d\ p \leq |(d\ t \cdot x)^\star \cdot ad\ t]\ (d\ q \cdot ad\ t) \Longrightarrow d\ p \leq |d\ t \cdot x]\ d\ q$

oops

end

end

1.18 Range and Antirange Semirings

theory *Range-Semiring*
imports *Antidomain-Semiring*
begin

1.18.1 Range Semirings

We set up the duality between domain and antidomain semirings on the one hand and range and antirange semirings on the other hand.

class *range-op* =
  fixes *range-op* :: $'a \Rightarrow 'a\ (r)$

class *range-semiring* = *semiring-one-zero* + *plus-ord* + *range-op* +
assumes *rsr1* [*simp*]: $x + (x \cdot r\ x) = x \cdot r\ x$
and *rsr2* [*simp*]: $r\ (r\ x \cdot y) = r(x \cdot y)$
and *rsr3* [*simp*]: $r\ x + 1 = 1$
and *rsr4* [*simp*]: $r\ 0 = 0$
and *rsr5* [*simp*]: $r\ (x + y) = r\ x + r\ y$

begin

definition *bd* :: $'a \Rightarrow 'a \Rightarrow 'a$ ($\langle$-$|$ - [*61*,*81*] *82*) **where**
$\langle x|\ y = r\ (y \cdot x)$

no-notation *range-op* (r)

end

sublocale *range-semiring* $\subseteq$ *rdual*: *domain-semiring* (+) $\lambda x\ y.\ y \cdot x$ *1 0 range-op* ($\leq$) ($<$)
rewrites *rdual.fd* $x\ y = \langle x|\ y$
proof –
show *class.domain-semiring* (+) ($\lambda x\ y.\ y \cdot x$) *1 0 range-op* ($\leq$) ($<$)
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *rdual*: *domain-semiring* (+) $\lambda x\ y.\ y \cdot x$ *1 0 range-op* ($\leq$) ($<$) **.**
show *rdual.fd* $x\ y = \langle x|\ y$
unfolding *rdual.fd-def bd-def* **by** *auto*
qed

sublocale *domain-semiring* $\subseteq$ *ddual*: *range-semiring* (+) $\lambda x\ y.\ y \cdot x$ *1 0 domain-op* ($\leq$) ($<$)
rewrites *ddual.bd* $x\ y$ = *domain-semiringl-class.fd* $x\ y$
proof –
show *class.range-semiring* (+) ($\lambda x\ y.\ y \cdot x$) *1 0 domain-op* ($\leq$) ($<$)
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *ddual*: *range-semiring* (+) $\lambda x\ y.\ y \cdot x$ *1 0 domain-op* ($\leq$) ($<$) **.**
show *ddual.bd* $x\ y$ = *domain-semiringl-class.fd* $x\ y$
unfolding *ddual.bd-def fd-def* **by** *auto*
qed

1.18.2 Antirange Semirings

class *antirange-op* =
fixes *antirange-op* :: $'a \Rightarrow 'a$ (*ar* - [*999*] *1000*)

class *antirange-semiring* = *semiring-one-zero* + *plus-ord* + *antirange-op* +
assumes *ars1* [*simp*]: $x \cdot ar\ x = 0$
and *ars2* [*simp*]: $ar\ (x \cdot y) + ar\ (ar\ ar\ x \cdot y) = ar\ (ar\ ar\ x \cdot y)$
and *ars3* [*simp*]: $ar\ ar\ x + ar\ x = 1$

begin

no-notation *bd* (⟨-| - [*61*,*81*] *82*)

definition *ars-r* :: ′*a* ⇒ ′*a* (*r*) **where**
  r x = *ar* (*ar x*)

definition *bdia* :: ′*a* ⇒ ′*a* ⇒ ′*a* (⟨-| - [*61*,*81*] *82*) **where**
  ⟨*x*| *y* = *ar ar* (*y* · *x*)

definition *bbox* :: ′*a* ⇒ ′*a* ⇒ ′*a* ([-| - [*61*,*81*] *82*) **where**
  [*x*| *y* = *ar* (*ar y* · *x*)

end

sublocale *antirange-semiring* ⊆ *ardual*: *antidomain-semiring antirange-op* (+) λ*x y*. *y* · *x* *1* *0* (≤) (<)
  rewrites *ardual.ads-d x* = *r x*
  and *ardual.fdia x y* = ⟨*x*| *y*
  and *ardual.fbox x y* = [*x*| *y*
proof −
  show *class.antidomain-semiring antirange-op* (+) (λ*x y*. *y* · *x*) *1* *0* (≤) (<)
    by (*standard*, *auto simp*: *mult-assoc distrib-left*)
  then interpret *ardual*: *antidomain-semiring antirange-op* (+) λ*x y*. *y* · *x* *1* *0* (≤) (<) **.**
  show *ardual.ads-d x* = *r x*
    by (*simp add*: *ardual.ads-d-def local.ars-r-def*)
  show *ardual.fdia x y* = ⟨*x*| *y*
    unfolding *ardual.fdia-def bdia-def* **by** *auto*
  show *ardual.fbox x y* = [*x*| *y*
    unfolding *ardual.fbox-def bbox-def* **by** *auto*
qed

context *antirange-semiring*

begin

sublocale *rs*: *range-semiring* (+) (·) *1* *0* λ*x*. *ar ar x* (≤) (<)
  by *unfold-locales*

end

sublocale *antidomain-semiring* ⊆ *addual*: *antirange-semiring* (+) λ*x y*. *y* · *x* *1* *0* *antidomain-op* (≤) (<)
  rewrites *addual.ars-r x* = *d x*
  and *addual.bdia x y* = |*x*⟩ *y*
  and *addual.bbox x y* = |*x*] *y*
proof −
  show *class.antirange-semiring* (+) (λ*x y*. *y* · *x*) *1* *0* *antidomain-op* (≤) (<)

by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *addual*: *antirange-semiring* (+) $\lambda x\ y.\ y \cdot x\ 1\ 0$ *antidomain-op* $(\leq)$ $(<)$ **.**
show *addual.ars-r* $x = d\ x$
by (*simp add*: *addual.ars-r-def local.ads-d-def*)
show *addual.bdia* $x\ y = |x\rangle\ y$
unfolding *addual.bdia-def fdia-def* **by** *auto*
show *addual.bbox* $x\ y = |x]\ y$
unfolding *addual.bbox-def fbox-def* **by** *auto*
qed

1.18.3 Antirange Kleene Algebras

class *antirange-kleene-algebra* = *antirange-semiring* + *kleene-algebra*

sublocale *antirange-kleene-algebra* $\subseteq$ *dual*: *antidomain-kleene-algebra antirange-op* (+) $\lambda x\ y.\ y \cdot x\ 1\ 0$ $(\leq)$ $(<)$ *star*
by (*standard*, *auto simp*: *local.star-inductr′ local.star-inductl*)

sublocale *antidomain-kleene-algebra* $\subseteq$ *dual*: *antirange-kleene-algebra* (+) $\lambda x\ y.\ y \cdot x\ 1\ 0$ $(\leq)$ $(<)$ *star antidomain-op*
by (*standard*, *simp-all add*: *star-inductr star-inductl*)

Hence all range theorems have been derived by duality in a generic way.

end

1.19 Modal Kleene Algebras

This section studies laws that relate antidomain and antirange semirings and Kleene algebra, notably Galois connections and conjugations as those studied in [**?**, **?**].

theory *Modal-Kleene-Algebra*
imports *Range-Semiring*
begin

class *modal-semiring* = *antidomain-semiring* + *antirange-semiring* +
assumes *domrange* [*simp*]: $d\ (r\ x) = r\ x$
and *rangedom* [*simp*]: $r\ (d\ x) = d\ x$

begin

These axioms force that the domain algebra and the range algebra coincide.

lemma *domrangefix*: $d\ x = x \longleftrightarrow r\ x = x$
by (*metis domrange rangedom*)

lemma *box-diamond-galois-1*:

assumes $d\ p = p$ **and** $d\ q = q$
shows $\langle x|\ p \leq q \longleftrightarrow p \leq |x]\ q$
proof –
  have $\langle x|\ p \leq q \longleftrightarrow p \cdot x \leq x \cdot q$
    by (*metis assms domrangefix local.ardual.ds.fdemodalisation2 local.ars-r-def*)
  thus *?thesis*
    by (*metis assms fbox-demodalisation3*)
qed

lemma *box-diamond-galois-2*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $|x\rangle\ p \leq q \longleftrightarrow p \leq [x|\ q$
proof –
  have $|x\rangle\ p \leq q \longleftrightarrow x \cdot p \leq q \cdot x$
    by (*metis assms local.ads-d-def local.ds.fdemodalisation2*)
  thus *?thesis*
    by (*metis assms domrangefix local.ardual.fbox-demodalisation3*)
qed

lemma *diamond-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $|x\rangle\ p \leq ad\ q \longleftrightarrow \langle x|\ q \leq ad\ p$
proof –
  have $|x\rangle\ p \leq ad\ q \longleftrightarrow x \cdot p \leq ad\ q \cdot x$
    by (*metis assms local.ads-d-def local.ds.fdemodalisation2*)
  also have ... $\longleftrightarrow q \cdot x \leq x \cdot ad\ p$
    by (*metis assms local.ads-d-def local.kat-1-equiv-opp*)
  finally show *?thesis*
   by (*metis assms box-diamond-galois-1 local.ads-d-def local.fbox-demodalisation3*)
qed

lemma *diamond-conjungation-aux*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $\langle x|\ p \leq ad\ q \longleftrightarrow q \cdot \langle x|\ p = 0$
apply *standard*
 apply (*metis assms*(*2*) *local.a-antitone′ local.a-gla local.ads-d-def*)
proof –
  assume *a1*: $q \cdot \langle x|\ p =$ *zero-class.zero*
  have $ad\ (ad\ q) = q$
    using *assms*(*2*) *local.ads-d-def* **by** *fastforce*
  then show $\langle x|\ p \leq ad\ q$
   using *a1* **by** (*metis* (*no-types*) *domrangefix local.a-gla local.ads-d-def local.antisym local.ardual.a-gla2 local.ardual.gla-1 local.ars-r-def local.bdia-def local.eq-refl*)
qed

lemma *diamond-conjugation*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $p \cdot |x\rangle\ q = 0 \longleftrightarrow q \cdot \langle x|\ p = 0$
proof –

have $p \cdot |x\rangle\ q = 0 \longleftrightarrow |x\rangle\ q \leq ad\ p$
by (*metis assms*(*1*) *local.a-gla local.ads-d-def local.am2 local.fdia-def*)
also have ... $\longleftrightarrow \langle x|\ p \leq ad\ q$
by (*simp add*: *assms diamond-conjugation-var-1*)
finally show *?thesis*
by (*simp add*: *assms diamond-conjungation-aux*)
qed

lemma *box-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $ad\ p \leq [x|\ q \longleftrightarrow ad\ q \leq |x]\ p$
by (*metis assms box-diamond-galois-1 box-diamond-galois-2 diamond-conjugation-var-1 local.ads-d-def*)

lemma *box-diamond-cancellation-1*: $d\ p = p \Longrightarrow p \leq |x]\ \langle x|\ p$
using *box-diamond-galois-1 domrangefix local.ars-r-def local.bdia-def* **by** *fastforce*

lemma *box-diamond-cancellation-2*: $d\ p = p \Longrightarrow p \leq [x|\ |x\rangle\ p$
by (*metis box-diamond-galois-2 local.ads-d-def local.dpdz.domain-invol local.eq-refl local.fdia-def*)

lemma *box-diamond-cancellation-3*: $d\ p = p \Longrightarrow |x\rangle\ [x|\ p \leq p$
using *box-diamond-galois-2 domrangefix local.ardual.fdia-fbox-de-morgan-2 local.ars-r-def local.bbox-def local.bdia-def* **by** *auto*

lemma *box-diamond-cancellation-4*: $d\ p = p \Longrightarrow \langle x|\ |x]\ p \leq p$
using *box-diamond-galois-1 local.ads-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2* **by** *auto*

end

class *modal-kleene-algebra* = *modal-semiring* + *kleene-algebra*
begin

subclass *antidomain-kleene-algebra* **..**

subclass *antirange-kleene-algebra* **..**

end

end

1.20 Models of Modal Kleene Algebras

theory *Modal-Kleene-Algebra-Models*
imports *Kleene-Algebra-Models Modal-Kleene-Algebra*

begin

This section develops the relation model. We also briefly develop the trace model for antidomain Kleene algebras, but not for antirange or full modal Kleene algebras. The reason is that traces are implemented as lists; we therefore expect tedious inductive proofs in the presence of range. The language model is not particularly interesting.

definition *rel-ad* :: *'a rel* ⇒ *'a rel* **where**
  rel-ad R = {(*x*,*x*) | *x*. ¬ (∃ *y*. (*x*,*y*) ∈ *R*)}

interpretation *rel-antidomain-kleene-algebra*: *antidomain-kleene-algebra*
   rel-ad (∪) (*O*) *Id* {} (⊆) (⊂) *rtrancl*
  by *unfold-locales* (*auto simp*: *rel-ad-def*)

definition *trace-a* :: (*'p*, *'a*) *trace set* ⇒ (*'p*, *'a*) *trace set* **where**
  trace-a X = {(*p*,[]) | *p*. ¬ (∃ *x*. *x* ∈ *X* ∧ *p* = *first x*)}

interpretation *trace-antidomain-kleene-algebra*: *antidomain-kleene-algebra trace-a* (∪) *t-prod t-one t-zero* (⊆) (⊂) *t-star*
proof
  show ⋀*x*. *t-prod* (*trace-a x*) *x* = *t-zero*
    by (*auto simp*: *trace-a-def t-prod-def t-fusion-def t-zero-def*)
  show ⋀*x y*. *trace-a* (*t-prod x y*) ∪ *trace-a* (*t-prod x* (*trace-a* (*trace-a y*))) = *trace-a* (*t-prod x* (*trace-a* (*trace-a y*)))
    by (*auto simp*: *trace-a-def t-prod-def t-fusion-def*)
  show ⋀*x*. *trace-a* (*trace-a x*) ∪ *trace-a x* = *t-one*
    by (*auto simp*: *trace-a-def t-one-def*)
qed

The trace model should be extended to cover modal Kleene algebras in the future.

definition *rel-ar* :: *'a rel* ⇒ *'a rel* **where**
  rel-ar R = {(*y*,*y*) | *y*. ¬ (∃ *x*. (*x*,*y*) ∈ *R*)}

interpretation *rel-antirange-kleene-algebra*: *antirange-semiring* (∪) (*O*) *Id* {} *rel-ar* (⊆) (⊂)
apply *unfold-locales*
apply (*simp-all add*: *rel-ar-def*)
by *auto*

interpretation *rel-modal-kleene-algebra*: *modal-kleene-algebra* (∪) (*O*) *Id* {} (⊆) (⊂) *rtrancl rel-ad rel-ar*
apply *standard*
apply (*metis* (*no-types*, *lifting*) *rel-antidomain-kleene-algebra.a-d-closed rel-antidomain-kleene-algebra.a-*
rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem
rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.apd-d-def rel-antidomain-kleene-alg
rel-antidomain-kleene-algebra.dpdz.dom-one rel-antirange-kleene-algebra.ardual.a-comm'
rel-antirange-kleene-algebra.ardual.a-d-closed rel-antirange-kleene-algebra.ardual.a-mul-d'
rel-antirange-kleene-algebra.ardual.a-mult-idem rel-antirange-kleene-algebra.ardual.a-zero
rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardual.am6-lem

rel-antirange-kleene-algebra.ardual.apd-d-def rel-antirange-kleene-algebra.ardual.s4)
by (*metis rel-antidomain-kleene-algebra.a-zero rel-antidomain-kleene-algebra.addual.ars1 rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.ds.ddual.mult-oner rel-antidomain-kleene-algebra.s4 rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardual.am rel-antirange-kleene-algebra.ardual.apd1 rel-antirange-kleene-algebra.ardual.dpdz.dns1''*)

end

1.21 Components Based on Kleene Algebra with Domain

theory *VC-KAD*

imports *Modal-Kleene-Algebra-Models Algebraic-VCs.P2S2R*

begin

1.21.1 Verification Component for Backward Reasoning

This component supports the verification of simple while programs in a partial correctness setting.

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

notation *p2r* ($\lceil$-$\rceil$)
notation *r2p* ($\lfloor$-$\rfloor$)

context *antidomain-kleene-algebra*
begin

Additional Facts for KAD

lemma *fbox-shunt*: $d\ p \cdot d\ q \leq |x]\ t \longleftrightarrow d\ p \leq ad\ q + |x]\ t$
by (*metis a-6 a-antitone' a-loc add-commute addual.ars-r-def am-d-def da-shunt2 fbox-def*)

Syntax for Conditionals and Loops

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

definition *while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - od* [*64*,*64*] *63*) **where**
while p do x od $= (d\ p \cdot x)^\star \cdot ad\ p$

definition *whilei* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*while - inv - do - od* [*64*,*64*,*64*] *63*) **where**
while p inv i do x od = *while p do x od*

Basic Weakest (Liberal) Precondition Calculus

In the setting of Kleene algebra with domain, the wlp operator is the forward modal box operator of antidomain Kleene algebra.

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-export2*: $|x]\ p \leq |x \cdot ad\ q]\ (d\ p \cdot ad\ q)$
proof –
  {fix t
   have $d\ t \cdot x \leq x \cdot d\ p \Longrightarrow d\ t \cdot x \cdot ad\ q \leq x \cdot ad\ q \cdot d\ p \cdot ad\ q$
    by (*metis* (*full-types*) *a-comm-var a-mult-idem ads-d-def am2 ds.ddual.mult-assoc phl-export2*)
   hence $d\ t \leq |x]\ p \Longrightarrow d\ t \leq |x \cdot ad\ q]\ (d\ p \cdot ad\ q)$
     by (*metis a-closure′ addual.ars-r-def ans-d-def dka.dsg3 ds.ddual.mult-assoc fbox-def fbox-demodalisation3*)
  }
  thus *?thesis*
   by (*metis a-closure′ addual.ars-r-def ans-d-def fbox-def order-refl*)
qed

lemma *fbox-export3*: $|x \cdot ad\ p]\ q = |x]\ (d\ p + d\ q)$
  using *a-de-morgan-var-3 ds.ddual.mult-assoc fbox-def* **by** *auto*

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

lemma *fbox-seq-var*: $p' \leq |y]\ q \Longrightarrow p \leq |x]\ p' \Longrightarrow p \leq |x \cdot y]\ q$
proof –
  assume *h1*: $p \leq |x]\ p'$ **and** *h2*: $p' \leq |y]\ q$
  hence $|x]\ p' \leq |x]\ |y]\ q$
   by (*simp add*: *dka.dom-iso fbox-iso*)
  thus *?thesis*
   by (*metis h1 dual-order.trans fbox-seq*)
qed

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

lemma *fbox-cond-aux1* [*simp*]: $d\ p \cdot |$*if p then x else y fi*$]\ q = d\ p \cdot |x]\ q$
proof –
  have $d\ p \cdot |$*if p then x else y fi*$]\ q = d\ p \cdot |x]\ q \cdot (d\ p + d\ (\ |y]\ q))$
   using *a-d-add-closure addual.ars-r-def ds.ddual.mult-assoc fbox-def fbox-cond-var* **by** *auto*
  thus *?thesis*
   by (*metis addual.ars-r-def am2 dka.dns5 ds.ddual.mult-assoc fbox-def*)
qed

lemma *fbox-cond-aux2* [*simp*]: *ad p* · |*if p then x else y fi*] *q* = *ad p* · |*y*] *q*
by (*metis cond-def a-closure′ add-commute addual.ars-r-def ans-d-def fbox-cond-aux1*)

lemma *fbox-cond* [*simp*]: |*if p then x else y fi*] *q* = (*d p* · |*x*] *q*) + (*ad p* · |*y*] *q*)
proof −
have |*if p then x else y fi*] *q* = (*d p* + *ad p*) · |*if p then x else y fi*] *q*
by (*simp add*: *addual.ars-r-def*)
thus *?thesis*
by (*metis distrib-right′ fbox-cond-aux1 fbox-cond-aux2*)
qed

lemma *fbox-cond-var2* [*simp*]: |*if p then x else y fi*] *q* = *if p then* |*x*] *q else* |*y*] *q fi*
using *cond-def fbox-cond* **by** *auto*

lemma *fbox-while-unfold*: |*while t do x od*] *p* = (*d t* + *d p*) · (*ad t* + |*x*] |*while t do x od*] *p*)
by (*metis fbox-export1 fbox-export3 dka.dom-add-closed fbox-star-unfold-var while-def*)

lemma *fbox-while-var1*: *d t* · |*while t do x od*] *p* = *d t* · |*x*] |*while t do x od*] *p*
by (*metis fbox-while-unfold a-mult-add ads-d-def dka.dns5 ds.ddual.mult-assoc join.sup-commute*)

lemma *fbox-while-var2*: *ad t* · |*while t do x od*] *p* ≤ *d p*
proof −
have *ad t* · |*while t do x od*] *p* = *ad t* · (*d t* + *d p*) · (*ad t* + |*x*] |*while t do x od*] *p*)
by (*metis fbox-while-unfold ds.ddual.mult-assoc*)
also have ... = *ad t* · *d p* · (*ad t* + |*x*] |*while t do x od*] *p*)
by (*metis a-de-morgan-var-3 addual.ars-r-def dka.dom-add-closed s4*)
also have ... ≤ *d p* · (*ad t* + |*x*] |*while t do x od*] *p*)
using *a-subid-aux1′ mult-isor* **by** *blast*
finally show *?thesis*
by (*metis a-de-morgan-var-3 a-mult-idem addual.ars-r-def ans4 dka.dsr5 dpdz.domain-1′′ dual-order.trans fbox-def*)
qed

lemma *fbox-while*: *d p* · *d t* ≤ |*x*] *p* ⟹ *d p* ≤ |*while t do x od*] (*d p* · *ad t*)
proof −
assume *d p* · *d t* ≤ |*x*] *p*
hence *d p* ≤ |*d t* · *x*] *p*
by (*simp add*: *fbox-export1 fbox-shunt*)
hence *d p* ≤ |(*d t* · *x*)*] *p*
by (*simp add*: *fbox-star-induct-var*)
thus *?thesis*
using *order-trans while-def fbox-export2* **by** *presburger*
qed

lemma *fbox-while-var*: *d p* = |*d t* · *x*] *p* ⟹ *d p* ≤ |*while t do x od*] (*d p* · *ad t*)
by (*metis eq-refl fbox-export1 fbox-shunt fbox-while*)

lemma *fbox-whilei*: $d\ p \le d\ i \Longrightarrow d\ i \cdot ad\ t \le d\ q \Longrightarrow d\ i \cdot d\ t \le |x]\ i \Longrightarrow d\ p \le |while\ t\ inv\ i\ do\ x\ od]\ q$
proof −
  assume *a1*: $d\ p \le d\ i$ **and** *a2*: $d\ i \cdot ad\ t \le d\ q$ **and** $d\ i \cdot d\ t \le |x]\ i$
  hence $d\ i \le |while\ t\ inv\ i\ do\ x\ od]\ (d\ i \cdot ad\ t)$
    by (*simp add*: *fbox-while whilei-def*)
  also have ... $\le |while\ t\ inv\ i\ do\ x\ od]\ q$
    using *a2 dka.dom-iso fbox-iso* **by** *fastforce*
  finally show *?thesis*
    using *a1 dual-order.trans* **by** *blast*
qed

The next rule is used for dealing with while loops after a series of sequential steps.

lemma *fbox-whilei-break*: $d\ p \le |y]\ i \Longrightarrow d\ i \cdot ad\ t \le d\ q \Longrightarrow d\ i \cdot d\ t \le |x]\ i \Longrightarrow d\ p \le |y \cdot (while\ t\ inv\ i\ do\ x\ od)]\ q$
  apply (*rule fbox-seq-var*, *rule fbox-whilei*, *simp-all*, *blast*)
  using *fbox-simp* **by** *auto*

Finally we derive a frame rule.

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

lemma *fbox-frame-var*: $d\ p \le |x]\ p \Longrightarrow d\ q \le |x]\ t \Longrightarrow d\ p \cdot d\ q \le |x]\ (d\ p \cdot d\ t)$
  using *fbox-frame fbox-demodalisation3 fbox-simp* **by** *auto*

end

Store and Assignment

type-synonym $'a\ store = string \Rightarrow 'a$

notation *rel-antidomain-kleene-algebra.fbox* (*wp*)
and *rel-antidomain-kleene-algebra.fdia* (*relfdia*)

definition *gets* :: $string \Rightarrow ('a\ store \Rightarrow 'a) \Rightarrow 'a\ store\ rel$ (- ::= - [70, 65] 61)
where
  $v ::= e = \{(s,s\ (v := e\ s))\ |s.\ True\}$

lemma *assign-prop*: $\lceil \lambda s.\ P\ (s\ (v := e\ s)) \rceil\ ;\ (v ::= e) = (v ::= e)\ ;\ \lceil P \rceil$
  by (*auto simp add*: *p2r-def gets-def*)

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

lemma *wp-assign-var* [*simp*]: $\lfloor wp\ (v ::= e)\ \lceil Q \rceil \rfloor = (\lambda s.\ Q\ (s\ (v := e\ s)))$
  by (*simp*, *auto simp*: *r2p-def p2r-def*)

lemma *wp-assign-det*: *wp* $(v ::= e)$ $\lceil Q\rceil$ = *relfdia* $(v ::= e)$ $\lceil Q\rceil$
by (*auto simp add*: *rel-antidomain-kleene-algebra.fbox-def rel-antidomain-kleene-algebra.fdia-def gets-def p2r-def rel-ad-def*, *fast*)

Simplifications

notation *rel-antidomain-kleene-algebra.ads-d* (*rdom*)

abbreviation *spec-sugar* :: *'a pred* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a pred* $\Rightarrow$ *bool* (*PRE - - POST -* [*64*,*64*,*64*] *63*) **where**
PRE P X POST Q $\equiv$ *rdom* $\lceil P\rceil \subseteq$ *wp X* $\lceil Q\rceil$

abbreviation *cond-sugar* :: *'a pred* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* (*IF - THEN - ELSE - FI* [*64*,*64*,*64*] *63*) **where**
IF P THEN X ELSE Y FI $\equiv$ *rel-antidomain-kleene-algebra.cond* $\lceil P\rceil$ *X Y*

abbreviation *whilei-sugar* :: *'a pred* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* (*WHILE - INV - DO - OD* [*64*,*64*,*64*] *63*) **where**
WHILE P INV I DO X OD $\equiv$ *rel-antidomain-kleene-algebra.whilei* $\lceil P\rceil$ $\lceil I\rceil$ *X*

lemma *d-p2r* [*simp*]: *rdom* $\lceil P\rceil = \lceil P\rceil$
by (*simp add*: *p2r-def rel-antidomain-kleene-algebra.ads-d-def rel-ad-def*)

lemma *p2r-conj-hom-var-symm* [*simp*]: $\lceil P\rceil$; $\lceil Q\rceil = \lceil P \sqcap Q\rceil$
by (*simp add*: *p2r-conj-hom-var*)

lemma *p2r-neg-hom* [*simp*]: *rel-ad* $\lceil P\rceil = \lceil - P\rceil$
by (*simp add*: *rel-ad-def p2r-def*)

lemma *wp-trafo*: $\lfloor wp\ X\ \lceil Q\rceil\rfloor = (\lambda s.\ \forall s'.\ (s,s') \in X \longrightarrow Q\ s')$
by (*auto simp*: *rel-antidomain-kleene-algebra.fbox-def rel-ad-def p2r-def r2p-def*)

lemma *wp-trafo-var*: $\lfloor wp\ X\ \lceil Q\rceil\rfloor\ s = (\forall s'.\ (s,s') \in X \longrightarrow Q\ s')$
by (*simp add*: *wp-trafo*)

lemma *wp-simp*: *rdom* $\lceil\lfloor wp\ X\ Q\rfloor\rceil = wp\ X\ Q$
by (*metis d-p2r rel-antidomain-kleene-algebra.a-subid' rel-antidomain-kleene-algebra.addual.bbox-def rpr*)

lemma *wp-simp-var* [*simp*]: *wp* $\lceil P\rceil$ $\lceil Q\rceil = \lceil - P \sqcup Q\rceil$
by (*simp add*: *rel-antidomain-kleene-algebra.fbox-def*)

lemma *impl-prop* [*simp*]: $\lceil P\rceil \subseteq \lceil Q\rceil \longleftrightarrow (\forall s.\ P\ s \longrightarrow Q\ s)$
by (*auto simp*: *p2r-def*)

lemma *p2r-eq-prop* [*simp*]: $\lceil P\rceil = \lceil Q\rceil \longleftrightarrow (\forall s.\ P\ s \longleftrightarrow Q\ s)$
by (*auto simp*: *p2r-def*)

lemma *impl-prop-var* [*simp*]: *rdom* $\lceil P \rceil \subseteq$ *rdom* $\lceil Q \rceil \longleftrightarrow (\forall s.\ P\ s \longrightarrow\ Q\ s)$
  by *simp*

lemma *p2r-eq-prop-var* [*simp*]: *rdom* $\lceil P \rceil =$ *rdom* $\lceil Q \rceil \longleftrightarrow (\forall s.\ P\ s \longleftrightarrow\ Q\ s)$
  by *simp*

lemma *wp-whilei*: $(\forall s.\ P\ s \longrightarrow I\ s) \Longrightarrow (\forall s.\ (I \sqcap -T)\ s \longrightarrow Q\ s) \Longrightarrow (\forall s.\ (I \sqcap T)\ s \longrightarrow \lfloor wp\ X\ \lceil I \rceil \rfloor\ s)$
$\Longrightarrow (\forall s.\ P\ s \longrightarrow \lfloor wp\ (\mathit{WHILE}\ T\ \mathit{INV}\ I\ \mathit{DO}\ X\ \mathit{OD})\ \lceil Q \rceil \rfloor\ s)$
  apply (*simp only*: *impl-prop-var*[*symmetric*] *wp-simp*)
  by (*rule rel-antidomain-kleene-algebra.fbox-whilei*, *simp-all*, *simp-all add*: *p2r-def*)

end

1.22 VC_diffKAD

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

begin

1.22.1 Stack Theories Preliminaries: VC_KAD and ODEs

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

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

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

This and more notation is explained by the following lemmata.

lemma shows $\lceil P \rceil = \{(s,\ s)\ |s.\ P\ s\}$
  and $\lfloor R \rfloor = (\lambda x.\ x \in r2s\ R)$
  and $r2s\ R = \{x\ |x.\ \exists\ y.\ (x,y) \in R\}$
  and $\pi_1\ (x,y) = x \wedge \pi_2\ (x,y) = y$
  and $\Delta^c{}_1\ R = \{(x,\ x)\ |x.\ \nexists y.\ (x,\ y) \in R\}$
  and $wp\ R\ Q = \Delta^c{}_1\ (R\ ;\ \Delta^c{}_1\ Q)$
  and $[x1,x2,x3,x4] \otimes [y1,y2] = [(x1,y1),(x2,y2)]$

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

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

Observe also, the following consequences and facts:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.22.2 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**
$(\partial\ x) = ''d[''@x@'']''$

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

proposition *vdiff-inj*:$(\partial\ x) = (\partial\ y) \Longrightarrow x = y$

by(*simp add*: *vdiff-def*)

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

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

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

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

(primed) dSolve preliminaries

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

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

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

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

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

abbreviation *state-list-cross-upd* ::*real store* ⇒ (*string* × (*real store* ⇒ *real*)) *list* ⇒

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

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

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

theorem *state-list-cross-upd-its-vars*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *its-var*: (*u*,*x*,*f*) ∈ *set* (*uInput* ⊗ *xfList*)
shows (*s*[*xfList*←*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* ∈ *X* ⟹ (*override-on f g X*)(*x* := *z*) = (*override-on f* (*g*(*x* := *z*)) *X*)
by (*rule ext*, *simp add*: *override-on-def*)

lemma *inductive-state-list-cross-upd-its-dvars*:
assumes ∃ *g*. (*s*[*xfTail*←*uTail*] *0*) = *override-on s g varDiffs*
and ∀ *xf*∈*set* (*xf* # *xfTail*). π_1 *xf* ∉ *varDiffs*
and ∀ *uxf*∈*set* (*u* # *uTail* ⊗ *xf* # *xfTail*). π_1 *uxf 0 s* = *s* (π_1 (π_2 *uxf*))

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

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

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

lemma *has-vderiv-on-cong-open-interval*:
assumes *gHyp*: $\forall\ \tau > 0.\ f\ \tau = g\ \tau$ **and** *tHyp*: $t>0$
and *fHyp*: $(f$ *has-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*. $\pi_1\ xf \notin$ *varDiffs*
and *solHyp2*: $\forall\, t{\geq}0.\ ((\lambda\tau.\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)\ x)$
has-vderiv-on $(\lambda\tau.\ f\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)))\ \{0..t\}$
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* $f{=}(\lambda\tau.\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)\ x)$ **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*: $\forall\ s.\ \forall\, g.\ \forall\, xf{\in}set\ ((x, f)\ \#\ xfs).\ \pi_2\ xf$ (*override-on s g varDiffs*)
$= \pi_2\ xf\ s$
and *eqLengths*:*length* $((x, f)\ \#\ xfs) =$ *length* $(u\ \#\ us)$
and *distinct*:*distinct* (*map* π_1 $((x, f)\ \#\ xfs))$
and *vars*: $\forall\, xf{\in}set\ ((x, f)\ \#\ xfs).\ \pi_1\ xf \notin$ *varDiffs*

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

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

lemma *inductive-to-sol-greater-than-zero-its-dvars*:
assumes *lengthHyp*:*length* ((*y*, *g*) # *xfs*) = *length* (*v* # *vs*)

and *distHyp*:*distinct* (*map* π_1 ((*y*, *g*) # *xfs*))
and *varHyp*: $\forall$ *xf*$\in$*set* ((*y*, *g*) # *xfs*). π_1 *xf* $\notin$ *varDiffs*
and *indHyp*:(*u*,*x*,*f*) $\in$ *set* (*vs* $\otimes$ *xfs*) $\Longrightarrow$ (*s*[*xfs*$\leftarrow$*vs*]*t*)(∂ *x*) = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2*$*_R$*t*} *t*
and *disjHyp*:(*v*, *y*, *g*) = (*u*, *x*, *f*) $\vee$ (*u*, *x*, *f*) $\in$ *set* (*vs* $\otimes$ *xfs*) **and** *tHyp*:*t* > *0*
shows (*s*[(*y*, *g*) # *xfs*$\leftarrow$*v* # *vs*] *t*) (∂ *x*) = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t*
proof−
let *?lhs* = ((*s*[*xfs*$\leftarrow$*vs*] *t*)(*y* := *v t s*, ∂ *y* := *vderiv-of* (λ *r*. *v r s*) {*0*<..< (*2* $\cdot$ *t*)} *t*)) (∂ *x*)
let *?rhs* = *vderiv-of* (λ*r*. *u r s*) {*0*<..< (*2* $\cdot$ *t*)} *t*
have (*s*[(*y*, *g*) # *xfs*$\leftarrow$*v* # *vs*] *t*) (∂ *x*) = *?lhs* **using** *tHyp* **by** *simp*
also have *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t* =*?rhs* **by** *simp*
ultimately have *obs*:*?thesis* = (*?lhs* = *?rhs*) **by** *simp*
from *disjHyp* **have** *?lhs* = *?rhs*
proof
  assume *uxfEq*:(*v*, *y*, *g*) = (*u*, *x*, *f*)
  then have *?lhs* = *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $\cdot$ *t*)} *t* **by** *simp*
  also have *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $\cdot$ *t*)} *t* = *?rhs* **using** *uxfEq* **by** *simp*
  ultimately show *?lhs* = *?rhs* **by** *simp*
next
  assume *sygTail*:(*u*, *x*, *f*) $\in$ *set* (*vs* $\otimes$ *xfs*)
  from *this* **have** *y* $\neq$ *x* **using** *distHyp set-zip-left-rightD* **by** *force*
  hence ∂ *x* $\neq$ ∂ *y* **by**(*simp add*: *vdiff-def*)
  have *y* $\neq$ ∂ *x* **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
  then have *?lhs* = (*s*[*xfs*$\leftarrow$*vs*] *t*) (∂ *x*) **using** ‹*y* $\neq$ ∂ *x*› **and** ‹∂ *x* $\neq$ ∂ *y*› **by** *simp*
  also have (*s*[*xfs*$\leftarrow$*vs*] *t*) (∂ *x*) = *?rhs* **using** *indHyp sygTail* **by** *simp*
  ultimately show *?lhs* = *?rhs* **by** *simp*
qed
from *this* **and** *obs* **show** *?thesis* **by** *simp*
qed

lemma *to-sol-greater-than-zero-its-dvars*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *uxfHyp*:(*u*, *x*, *f*) $\in$ *set* (*uInput* $\otimes$ *xfList*) **and** *tHyp*:*t* > *0*
shows (*s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ *x*) = *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $*_R$ *t*)} *t*
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*, *clarify*)
by(*rule-tac f*=*f* **in** *inductive-to-sol-greater-than-zero-its-dvars*, *auto*)

dInv preliminaries

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

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

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

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

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

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

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

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

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

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

$xtList\langle\vartheta \oplus \eta\rangle = (xtList\langle\vartheta\rangle) \oplus (xtList\langle\eta\rangle)|$
$xtList\langle\vartheta \odot \eta\rangle = (xtList\langle\vartheta\rangle) \odot (xtList\langle\eta\rangle)$

proposition *substList-on-compl-of-varDiffs*:
assumes $trmVars\ \eta \subseteq (UNIV - varDiffs)$
and $set\ (map\ \pi_1\ xtList) \subseteq varDiffs$
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* ⇒ *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* × *trms*) *list* ⇒ *props* ⇒ *props* (-↾-↾ [*54*] *80*) **where**
$xtList\restriction\vartheta \doteq \eta\restriction = ((xtList\langle\vartheta\rangle) \doteq (xtList\langle\eta\rangle))|$
$xtList\restriction\vartheta \prec \eta\restriction = ((xtList\langle\vartheta\rangle) \prec (xtList\langle\eta\rangle))|$
$xtList\restriction\vartheta \preceq \eta\restriction = ((xtList\langle\vartheta\rangle) \preceq (xtList\langle\eta\rangle))|$
$xtList\restriction\varphi \sqcap \psi\restriction = ((xtList\restriction\varphi\restriction) \sqcap (xtList\restriction\psi\restriction))|$
$xtList\restriction\varphi \sqcup \psi\restriction = ((xtList\restriction\varphi\restriction) \sqcup (xtList\restriction\psi\restriction))$

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*=λ *t*. *2* $\cdot$ *x* $\cdot$ *t* **in** *derivative-eq-intros*(*6*))
apply(*rule-tac f'1*=λ *t*. *t* **in** *derivative-eq-intros*(*15*))
by (*auto intro*: *derivative-eq-intros*)

lemma *quadratic-monomial-derivative2*:
$((\lambda t{::}real.\ a \cdot t^2 / 2)$ *has-derivative* $(\lambda t.\ a \cdot x \cdot t))$ (*at x within T*)
apply(*rule-tac f'1*=$\lambda t.\ a \cdot (2 \cdot x \cdot t)$ **and** *g'1*=λ *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'*=λ *x*. $a \cdot x + v$ **and** *g'1*=λ *x*. *0* **in** *derivative-intros*(*191*))
apply(*rule-tac f'1*=λ *x*. $a \cdot x$ **and** *g'1*=λ *x*. *v* **in** *derivative-intros*(*191*))
using *poly-deriv*(*2*) **by**(*auto intro*: *derivative-intros*)

lemma [*poly-deriv*]:
$t \in T \Longrightarrow ((\lambda\tau.\ a \cdot \tau^2 / 2 + v \cdot \tau + x)$ *has-derivative* $(\lambda x.\ x *_R (a \cdot t + v)))$

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

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

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

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

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

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

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

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

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

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

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

begin

1.22.3 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 $y \notin (\pi_1 \langle\!\langle set\ xfList \rangle\!\rangle)$
shows $\varphi_S\ t\ (\partial\ y) = 0$
using *assms solves-store-ivpE*(*2*) **by** *simp*

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

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

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

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

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

1.22.4 Derivation of Differential Dynamic Logic Rules

"Differential Weakening"

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

theorem *dWeakening*:
assumes *guardImpliesPost*: $\lceil G \rceil \subseteq \lceil Q \rceil$
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*

using *assms* **and** *wlp-evol-guard* **by** (*metis* (*no-types*, *hide-lams*) *d-p2r order-trans p2r-subid rel-antidomain-kleene-algebra.fbox-iso*)

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

"Differential Cut"

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

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

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

thus $Q\ y$ **using** $*$ **by** *auto*
qed

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

Solve Differential Equation

lemma *prelim-dSolve*:
assumes *solHyp*:($\lambda t.$ *sol s*[*xfList*$\leftarrow$*uInput*] *t*) *solvesTheStoreIVP xfList withInitState s*
and *uniqHyp*:$\forall\ X.$ *solvesStoreIVP X xfList s* $\longrightarrow (\forall\ t \geq 0.$ (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) $= X\ t)$
and *diffAssgn*: $\forall\ t{\geq}0.\ G$ (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) $\longrightarrow Q$ (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)
shows $\forall\ c.\ (s,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow Q\ c$
proof(*clarify*)
fix c **assume** $(s,c) \in (ODEsystem\ xfList\ with\ G)$
from *this* **obtain** $t{::}real$ **and** $\varphi_S{::}real \Rightarrow real\ store$
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*) ⌈*Q*⌉ =
⌈λ *s*. ∀ *t*≥*0*. (∀ *r*∈{*0*..*t*}. *G* (*sol s*[*xfList*←*uInput*] *r*)) ⟶ *Q* (*sol s*[*xfList*←*uInput*] *t*)⌉
apply(*simp add*: *p2r-def*, *rule subset-antisym*)
unfolding *guarDiffEqtn-def rel-antidomain-kleene-algebra.fbox-def rel-ad-def*
using *solHyp* **apply**(*simp add*: *relcomp.simps*) **apply** *clarify*
apply(*rule-tac x*=*x* **in** *exI*, *clarsimp*)
apply(*erule-tac x*=*sol x*[*xfList*←*uInput*] *t* **in** *allE*, *erule disjE*)
apply(*erule-tac x*=*x* **in** *allE*, *erule-tac x*=*t* **in** *allE*)
apply(*erule impE*, *simp*, *erule-tac x*=λ*t*. *sol x*[*xfList*←*uInput*] *t* **in** *allE*)
apply(*simp-all*, *clarify*, *rule-tac x*=*s* **in** *exI*, *simp add*: *relcomp.simps*)
using *uniqHyp* **by** *fastforce*

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

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

lemma *conds4vdiffs-prelim*:
assumes *funcsHyp*:∀ *s g*. ∀ *xf*∈*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:∀ *uxf* ∈ *set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:∀ *t*≥*0*. ((λτ. (*sol s*[*xfList*←*uInput*] τ) *x*)
has-vderiv-on (λτ. *f* (*sol s*[*xfList*←*uInput*] τ))) {*0*..*t*}
and *xfHyp*:(*x*, *f*) ∈ *set xfList* **and** *tHyp*:*t* ≥ *0*
shows (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *t*)
proof−
from *xfHyp* **obtain** *u* **where** *xfuHyp*: (*u*,*x*,*f*) ∈ *set* (*uInput* ⊗ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
show (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) =*f* (*sol s*[*xfList*←*uInput*] *t*)
  proof(*cases t*=*0*)
  case *True*

have (*sol s*[*xfList*←*uInput*] *0*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *0*)
using *assms* **and** *to-sol-zero-its-dvars* **by** *blast*
then show *?thesis* **using** *True* **by** *blast*
next
case *False*
from *this* **have** *t* > *0* **using** *tHyp* **by** *simp*
hence (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *vderiv-of* (λ *r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t*
using *xfuHyp assms to-sol-greater-than-zero-its-dvars* **by** *blast*
also have *vderiv-of* (λ*r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t* = *f* (*sol s*[*xfList*←*uInput*] *t*)
using *assms xfuHyp* ‹*t* > *0*› **and** *vderiv-of-to-sol-its-vars* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
qed

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

lemma *conds4Consts*:
assumes *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
shows ∀ *x*. *x* ∉ (π_1⦇*set xfList*⦈) ⟶ (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *0*
using *varsHyp* **apply**(*induct xfList uInput rule*: *list-induct2'*)
apply(*simp-all add*: *override-on-def varDiffs-def vdiff-def*)
by *clarsimp*

lemma *conds4InitState*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:∀ *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 *xfHyp*:(*x*, *f*) ∈ *set xfList*
shows (*sol s*[*xfList*←*uInput*] *0*) *x* = *s x*
proof−
from *xfHyp* **obtain** *u* **where** *uxfHyp*:(*u*, *x*, *f*) ∈ *set* (*uInput* ⊗ *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*←*uInput*] *0*) *x* = *u 0* (*sol s*)
using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*
ultimately show (*sol s*[*xfList*←*uInput*] *0*) *x* = *s x* **using** *toZeroHyp* **by** *simp*
qed

lemma *conds4RestOfStrings*:
assumes $x \notin (\pi_1$(|*set xfList*|)) ∪ *varDiffs*
shows (*sol s*[*xfList*←*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*:∀ *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 *solHyp2*:∀ *t* ≥ *0*. ∀ *xf* ∈ *set xfList*.
((λ*t*. (*sol s*[*xfList*←*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λ*t*. π_2 *xf* (*sol s*[*xfList*←*uInput*] *t*))) {*0*..*t*}
shows *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*←*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*:∀ *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*:∀ *s*.∀ *uxf*∈*set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:∀ *s*.∀ *t* ≥ *0*. ∀ *xf* ∈ *set xfList*.
((λ*t*. (*sol s*[*xfList*←*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λ*t*. π_2 *xf* (*sol s*[*xfList*←*uInput*] *t*))) {*0*..*t*}
and *uniqHyp*:∀ *s*.∀ *X*. *solvesStoreIVP X xfList s* ⟶ (∀ *t* ≥ *0*. (*sol s*[*xfList*←*uInput*] *t*) = *X t*)
and *postCondHyp*:∀ *s*. *P s* ⟶ (∀ *t*≥*0*. *Q* (*sol s*[*xfList*←*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* ⇒ *real*
assumes *tHyp*:$t \geq 0$
and *contHyp*:*continuous-on* ({*0*..*t*} × *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* ∉ *varDiffs* ⟶ 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*:∀ *xf* ∈ *set xfList*. *continuous-on* ({*0*..*t*} × *UNIV*) (λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*←*uInput*] *t*))
and *eqDerivs*:∀ *xf* ∈ *set xfList*. ∀ τ ∈ {*0*..*t*}. (π_2 *xf*) (φ_s τ) = (π_2 *xf*) (*sol s*[*xfList*←*uInput*] τ)
and *Fsolves*:*solvesStoreIVP* φ_s *xfList s*
and *solHyp*:*solvesStoreIVP* (λ τ. (*sol s*[*xfList*←*uInput*] τ)) *xfList s*
shows (*sol s*[*xfList*←*uInput*] *t*) = φ_s *t*
proof
  fix *x*::*string* **show** (*sol s*[*xfList*←*uInput*] *t*) *x* = φ_s *t x*
  proof(*cases x* ∈ (π_1(|*set xfList*|)) ∪ *varDiffs*)
  case *False*
    then have *notInVars*:*x* ∉ (π_1(|*set xfList*|)) ∪ *varDiffs* **by** *simp*
    from *solHyp* **have** (*sol s*[*xfList*←*uInput*] *t*) *x* = *s x*
    using *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
   also from *Fsolves* **have** φ_s *t x* = *s x* **using** *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
    ultimately show (*sol s*[*xfList*←*uInput*] *t*) *x* = φ_s *t x* **by** *simp*
  next case *True*
    then have *x* ∈ (π_1(|*set xfList*|)) ∨ *x* ∈ *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 φ_s *t x = 0*
using *xDef solves-store-ivpD(2) Fsolves tHyp* **by** *simp*
also have *(sol s[xfList←uInput] t) x = 0*
using *False solHyp tHyp solves-store-ivpD(2) xDef* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed

theorem *dSolveUBC*:
assumes *contHyp*:$\forall$ *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*←*uInput*] *t*))
and *solHyp*:$\forall$ *s*. *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*←*uInput*] *t*)) *xfList s*
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*←*uInput*] *r*))
and *diffAssgn*: $\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *G* (*sol s*[*xfList*←*uInput*] *t*) $\longrightarrow$ *Q* (*sol s*[*xfList*←*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput=uInput* **in** *dSolve*)
prefer *2* **subgoal proof**(*clarify*)
fix *s*::*real store* **and** φ_s::*real* $\Rightarrow$ *real store* **and** *t*::*real*
assume *isSol*:*solvesStoreIVP* φ_s *xfList s* **and** *sHyp*:*0* $\leq$ *t*
from *this* **and** *uniqHyp* **have** $\forall$ *xf* $\in$ *set xfList*. $\forall$ *t* $\in$ $\{0..t\}$.
(π_2 *xf*) (φ_s *t*) = (π_2 *xf*) (*sol s*[*xfList*←*uInput*] *t*) **by** *auto*
also have $\forall$ *xf* $\in$ *set xfList*. *continuous-on* ($\{0..t\}$ $\times$ *UNIV*)
(λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*←*uInput*] *t*)) **using** *contHyp sHyp* **by** *blast*
ultimately show (*sol s*[*xfList*←*uInput*] *t*) = φ_s *t*
using *sHyp isSol ubcStoreUniqueSol solHyp* **by** *simp*
qed using *assms* **by** *simp-all*

theorem *dSolve-toSolveUBC*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf* *s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *s*. $\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). π_1 *uxf 0* (*sol s*) = *sol s* (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *s*. $\forall$ *t*$\geq$*0*. $\forall$ *xf*$\in$*set xfList*. ((λ*t*. (*sol s*[*xfList*←*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on*
(λ*t*. π_2 *xf* (*sol s*[*xfList*←*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*←*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*←*uInput*]

r))
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolveUBC*)
using *contHyp* **apply** *simp*
apply(*rule allI*, *rule-tac uInput*=*uInput* **in** *conds4storeIVP-on-toSol*)
using *assms* **by** *auto*

"Differential Invariant."

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

lemma *derivationLemma-baseCase*:
fixes *F*::*real* $\Rightarrow$ *real store*
assumes *solves*:*solvesStoreIVP F xfList a*
shows $\forall$ *x* $\in$ (*UNIV* $-$ *varDiffs*). $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0*..*t*}.
((λ *t*. *F t x*) *has-vector-derivative F r* (∂ *x*)) (*at r within* {*0*..*t*})
proof
fix *x*
assume *x* $\in$ *UNIV* $-$ *varDiffs*
then have *notVarDiff*:$\forall$ *z*. *x* $\neq$ ∂ *z* **using** *varDiffs-def* **by** *fastforce*
 show $\forall$ *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(*cases x* $\in$ *set* (*map* π_1 *xfList*))
   case *True*
   from *this* **and** *solves* **have** $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0*..*t*}. $\forall$ *xf* $\in$ *set xfList*.
   ((λ *t*. *F t* (π_1 *xf*)) *has-vector-derivative F r* (∂ (π_1 *xf*))) (*at r within* {*0*..*t*})
   apply(*rule-tac solvesStoreIVP-couldBeModified*) **using** *solves solves-store-ivpD*

by *auto*
  from *this* **show** *?thesis* **using** *True* **by** *auto*
 next
  case *False*
  from *this notVarDiff* **and** *solves* **have** *const*:$\forall\ t \geq 0.\ F\ t\ x = a\ x$
  using *solves-store-ivpD*(*1*) **by** (*simp add*: *varDiffs-def*)
  have *constD*:$\forall\ t \geq 0.\ \forall r\in\{0..t\}.\ ((\lambda\ r.\ a\ x)$ *has-vector-derivative 0*) (*at r within* $\{0..t\}$)
  by (*auto intro*: *derivative-eq-intros*)
  {fix *t r*::*real*
   assume $t\geq 0$ **and** $r \in \{0..t\}$
   hence ((λ *s. a x*) *has-vector-derivative 0*) (*at r within* $\{0..t\}$) **by** (*simp add*: *constD*)
   moreover have $\bigwedge s.\ s \in \{0..t\} \Longrightarrow (\lambda\ r.\ F\ r\ x)\ s = (\lambda\ r.\ a\ x)\ s$
   using *const* **by** (*simp add*: ‹$0 \leq t$›)
   ultimately have ((λ *s. F s x*) *has-vector-derivative 0*) (*at r within* $\{0..t\}$)
   using *has-vector-derivative-transform* **by** (*metis* ‹$r \in \{0..t\}$›)**}**
  hence *isZero*:$\forall t\geq 0.\forall r\in\{0..t\}.((\lambda\ t.\ F\ t\ x)$*has-vector-derivative 0*)(*at r within* $\{0..t\}$)**by** *blast*
  from *False solves* **and** *notVarDiff* **have** $\forall\ t \geq 0.\ F\ t\ (\partial\ x) = 0$
  using *solves-store-ivpD*(*2*) **by** *simp*
  then show *?thesis* **using** *isZero* **by** *simp*
 qed
qed

lemma *derivationLemma*:
assumes *solvesStoreIVP F xfList a*
and *tHyp*:$t \geq 0$
and *termVarsHyp*:$\forall\ x \in$ *trmVars* $\eta.\ x \in (UNIV - varDiffs)$
shows $\forall r\in\{0..t\}.\ ((\lambda\ s.\ [\![\eta]\!]_t\ (F\ s))$*has-vector-derivative* $[\![\partial_t\ \eta]\!]_t\ (F\ r))$ (*at r within* $\{0..t\}$)
using *termVarsHyp* **proof**(*induction* 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* η*1* η*2*)
then show *?case*
apply(*clarsimp*)
apply(*subgoal-tac* (($\lambda s.$ $[\![\eta 1]\!]_t$ $(F\ s)$ $*_R$ $[\![\eta 2]\!]_t$ $(F\ s)$) *has-vector-derivative* $[\![\partial_t\ \eta 1]\!]_t$ $(F\ r)$ $\cdot$ $[\![\eta 2]\!]_t$ $(F\ r)$ $+$ $[\![\eta 1]\!]_t$ $(F\ r)$ $\cdot$ $[\![\partial_t\ \eta 2]\!]_t$ $(F\ r)$) (*at r within* $\{0..t\}$),*simp*)
apply(*rule-tac* $f'1$=$[\![\partial_t\ \eta 1]\!]_t$ $(F\ r)$ **and** $g'1$=$[\![\partial_t\ \eta 2]\!]_t$ $(F\ r)$ **in** *derivative-eq-intros*(*25*))
by (*simp-all add*: *has-field-derivative-iff-has-vector-derivative*)
qed

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

ultimately show *?thesis* **by** *blast*
next
case *False*
then have $((map\ (vdiff \circ \pi_1)\ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V\ (\partial\ x)\rangle$
$= t_V\ (\partial\ x)$
using *substList-cross-vdiff-on-non-ocurring-var* **by**(*metis*(*no-types*, *lifting*) *List.map.compositionality*)
thus *?thesis* **by** *simp*
qed
qed

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

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

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

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

$\{0..t\}$)
using *obs2* **by** *auto*
from *this* **and** *tcHyp* **have** $\forall\, s{\in}\{0..t\}.\ ((\lambda x.\ [\![\eta]\!]_t\ (F\ x))$ *has-derivative* $(\lambda x.\ x *_R 0))$
(*at s within* $\{0..t\}$) **by** (*metis has-vector-derivative-def*)
hence $[\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = (\lambda x.\ x *_R 0)\ (t - 0)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then show $[\![\eta]\!]_t\ c = 0$ **using** *obs1 tcHyp* **by** *auto*
qed

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

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

lemma *dInvForProps-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ st \geq 0$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a > 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c > 0)$
and $[\![\eta]\!]_t\ a \geq 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c \geq 0)$
proof(*clarify*)

fix *c* **assume** *aHyp*:$[\![\eta]\!]_t$ *a* > *0* **and** *cHyp*:(*a*, *c*) ∈ *ODEsystem xfList with G*
from *this* **obtain** *t*::*real* **and** *F*::*real* ⇒ *real store*
where *tcHyp*:*t*≥*0* ∧ *F t* = *c* ∧ *solvesStoreIVP F xfList a* ∧ (∀ *r*∈{*0..t*}. *G* (*F r*))

using *guarDiffEqtn-def* **by** *auto*
then have ∀ *x*. *x* ∉ *varDiffs* ⟶ *F 0 x* = *a x* **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t$ *a* = $[\![\eta]\!]_t$ (*F 0*) **using** *termVarsHyp eqInVars-impl-eqInTrms*
by *blast*
hence *obs1*:$[\![\eta]\!]_t$ (*F 0*) > *0* **using** *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:∀ *r*∈{*0..t*}. ((λ*s*. $[\![\eta]\!]_t$ (*F s*)) *has-vector-derivative*
$[\![\partial_t\ \eta]\!]_t$ (*F r*)) (*at r within* {*0..t*}) **using** *derivationLemma termVarsHyp* **by** *blast*
have (∀ *t*≥*0*. ∀ *xf* ∈ *set xfList*. *F t* (∂ (π_1 *xf*)) = π_2 *xf* (*F t*))
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*
hence ∀ *r*∈{*0..t*}. $[\![\partial_t\ \eta]\!]_t$ (*F r*) = $[\![$((*map* (*vdiff* ∘ π_1) *xfList*) ⊗ *uInput*) ⟨∂_t η⟩$]\!]_t$
(*F r*)
using *diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** ∀ *r*∈{*0..t*}. $[\![$((*map* (*vdiff* ∘ π_1) *xfList*) ⊗ *uInput*) ⟨∂_t
η⟩$]\!]_t$ (*F r*) ≥ *0*
using *solves-store-ivpD*(*2*) *tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have ∗:∀ *r*∈{*0..t*}. $[\![\partial_t\ \eta]\!]_t$ (*F r*) ≥ *0* **by** (*simp*)
from *obs2* **and** *tcHyp* **have** ∀ *r*∈{*0..t*}. ((λ*s*. $[\![\eta]\!]_t$ (*F s*)) *has-derivative*
(λ*x*. *x* $*_R$ ($[\![\partial_t\ \eta]\!]_t$ (*F r*)))) (*at r within* {*0..t*}) **by** (*simp add*: *has-vector-derivative-def*)

hence ∃ *r*∈{*0..t*}. $[\![\eta]\!]_t$ (*F t*) − $[\![\eta]\!]_t$ (*F 0*) = *t* · ($[\![(\partial_t\ \eta)]\!]_t$) (*F r*)
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain *r* **where** $[\![\partial_t\ \eta]\!]_t$ (*F r*) ≥ *0* ∧ *0* ≤ *r* ∧ *r* ≤ *t* ∧ $[\![\partial_t\ \eta]\!]_t$ (*F t*) ≥ *0*
∧ $[\![\eta]\!]_t$ (*F t*) − $[\![\eta]\!]_t$ (*F 0*) = *t* · ($[\![\partial_t\ \eta]\!]_t$ (*F r*))
using ∗ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t$ *c* > *0*
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

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

using *guarDiffEqtn-def* **by** *auto*
then have ∀ *x*. *x* ∉ *varDiffs* ⟶ *F 0 x* = *a x* **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t$ *a* = $[\![\eta]\!]_t$ (*F 0*) **using** *termVarsHyp eqInVars-impl-eqInTrms*
by *blast*
hence *obs1*:$[\![\eta]\!]_t$ (*F 0*) ≥ *0* **using** *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:∀ *r*∈{*0..t*}. ((λ*s*. $[\![\eta]\!]_t$ (*F s*)) *has-vector-derivative*
$[\![\partial_t\ \eta]\!]_t$ (*F r*)) (*at r within* {*0..t*}) **using** *derivationLemma termVarsHyp* **by** *blast*
have (∀ *t*≥*0*. ∀ *xf* ∈ *set xfList*. *F t* (∂ (π_1 *xf*)) = π_2 *xf* (*F t*))
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*

from *this* **and** *tcHyp* **have** $\forall\, r\in\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) =$
$[\![((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall\, r\in\{0..t\}.\ [\![((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*{:}\forall\, r\in\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$ **by** (*simp*)
from *obs2* **and** *tcHyp* **have** $\forall\, r\in\{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-derivative*
$(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))$ (*at r within* $\{0..t\}$) **by** (*simp add*: *has-vector-derivative-def*)

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

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

lemma *less-pval-to-tval*:
assumes $[\![((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)\upharpoonleft\partial_P\ (\vartheta \prec \eta)\upharpoonleft]\!]_P\ st$
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 $[\![((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)\upharpoonleft\partial_P\ (\vartheta \preceq \eta)\upharpoonleft]\!]_P\ st$
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*:$\forall\ st.\ G\ st \longrightarrow\ (\forall\, str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) =$
$0) \longrightarrow$
$[\![((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput)\upharpoonleft\partial_P\ \varphi\upharpoonleft]\!]_P\ st$
and *propVarsHyp*:*propVars* $\varphi \subseteq (UNIV\ -\ varDiffs)$
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
shows $[\![\varphi]\!]_P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\varphi]\!]_P\ c)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\varphi]\!]_P\ a$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** *t*::*real* **and** *F*::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t\geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a$ **using** *guarDiffEqtn-def*
by *auto*
from *aHyp propVarsHyp* **and** *substHyp* **show** $[\![\varphi]\!]_P\ c$
proof(*induction* φ)
case (*Eq* $\vartheta\ \eta$)
hence *hyp*:$\forall\, st.\ G\ st \longrightarrow\ (\forall\, str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$

$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ (\vartheta \doteq \eta)\restriction]\!]_P\ st$ **by** *blast*
then have $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t\ (\vartheta \oplus (\ominus\ \eta))\rangle]\!]_t\ st = 0$ **by** *simp*
also have $trmVars\ (\vartheta \oplus (\ominus\ \eta)) \subseteq UNIV - varDiffs$ **using** *Eq.prems(2)* **by** *simp*
moreover have $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ a = 0$ **using** *Eq.prems(1)* **by** *simp*
ultimately have $(\forall\, c.\ (a, c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ c = 0)$
using *dInvForTrms-prelim listsHyp* **by** *blast*
hence $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ (F\ t) = 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t)$ **by** *simp*
also have $([\![\vartheta \doteq \eta]\!]_P)\ c = ([\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(Less\ \vartheta\ \eta)$
hence $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$
using *less-pval-to-tval* **by** *metis*
also from *Less.prems(2)***have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$ **by** *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a > 0$ **using** *Less.prems(1)* **by** *simp*
ultimately have $(\forall\, c.\ (a, c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c > 0)$
using *dInvForProps-prelim(1) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) > 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\eta]\!]_t\ (F\ t) > [\![\vartheta]\!]_t\ (F\ t)$ **by** *simp*
also have $[\![\vartheta \prec \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) < [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(Leq\ \vartheta\ \eta)$
hence $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$ **using** *leq-pval-to-tval*
by *metis*
also from *Leq.prems(2)***have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$ **by** *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a \geq 0$ **using** *Leq.prems(1)* **by** *simp*
ultimately have $(\forall\, c.\ (a, c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c \geq 0)$
using *dInvForProps-prelim(2) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) \geq 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $([\![\eta]\!]_t\ (F\ t) \geq [\![\vartheta]\!]_t\ (F\ t))$ **by** *simp*
also have $[\![\vartheta \preceq \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) \leq [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(And\ \varphi 1\ \varphi 2)$
then show *?case* **by**(*simp*)
next
case $(Or\ \varphi 1\ \varphi 2)$
from *this* **show** *?case* **by** *auto*
qed
qed

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

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

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

begin

1.22.5 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-trafo vdiff-def add-strict-increasing2*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
prefer *2* **apply**(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*)
apply(*clarify*, *rule-tac* $f'1=\lambda\ x.\ s\ ''v''$ **and** $g'1=\lambda\ x.\ 0$ **in** *derivative-intros*(*191*))
apply(*rule-tac* $f'1=\lambda\ x.0$ **and** $g'1=\lambda\ x.1$ **in** *derivative-intros*(*194*))
by(*auto intro*: *derivative-intros*)

Same hybrid program verified with dSolve and the system of ODEs: $x' = v, v' = a$. The uniqueness part of the proof requires a preliminary lemma.

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

lemma *motion-with-constant-acceleration*:
PRE $(\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]\ with\ (\lambda\ s.\ True))$
POST $(\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac uInput*=$[\lambda\ t\ s.\ s\ ''a'' \cdot t\ \hat{}\ 2/2 + s\ ''v'' \cdot t + s\ ''x'',$

λ *t s. s ''a''* $\cdot$ *t* + *s ''v''*] **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal by**(*simp add*: *wp-trafo vdiff-def add-strict-increasing2*)
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *clarify*, *rule conjI*)
by(*rule galilean-transform*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
by(*rule continuous-intros*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
subgoal for *s* φ_s *t r* **apply**(*rule flow-vel-is-galilean-vel*[*of* φ_s *''x''* - - - - *t*])
by(*simp-all add*: *varDiffs-def vdiff-def*)
apply(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*) **done**
by(*auto simp*: *varDiffs-def vdiff-def*)

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

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

lemma *single-hop-ball*:
PRE (λ *s. 0* $\leq$ *s ''x''* $\wedge$ *s ''x''* = *H* $\wedge$ *s ''v''* = *0* $\wedge$ *s ''g''* > *0* $\wedge$ *1* $\geq$ *c* $\wedge$ *c*

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

— Example of system proved with a differential invariant.
lemma *circular-motion*:
PRE (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'', $\ominus$ (t_V ''x'')] **in** *dInvForTrms*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*clarsimp*, *erule-tac x*=''r'' **in** *allE*)
by *simp*

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

— We revisit the two modes example from before, and prove it with invariants.
lemma *single-hop-ball-and-invariants*:
PRE (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'')) *FI*))
POST (λ *s*. *0* $\leq$ *s* ''x'' $\wedge$ *s* ''x'' $\leq$ *H*)
apply(*simp add*: *d-p2r*, *subgoal-tac rdom* $\lceil\lambda$*s*. *0* $\leq$ *s* ''x'' $\wedge$ *s* ''x'' = *H* $\wedge$ *s* ''v'' = *0* $\wedge$ *0* < *s* ''g'' $\wedge$ *c* $\leq$ *1* $\wedge$ *0* $\leq$ *c*$\rceil$

$\subseteq$ *wp* (*ODEsystem* [(''x'', $\lambda s.$ *s* ''v''), (''v'', $\lambda s.$ $-$ *s* ''g'')] *with* ($\lambda s.$ *0* $\leq$ *s* ''x'')
)
$\lceil$*inf* (*sup* ($-$ ($\lambda s.$ *s* ''x'' = *0*)) ($\lambda s.$ *0* $\leq$ *s* ''x'' $\wedge$ *s* ''x'' $\leq$ *H*)) (*sup* ($\lambda s.$ *s* ''x'' = *0*) ($\lambda s.$ *0* $\leq$ *s* ''x'' $\wedge$ *s* ''x'' $\leq$ *H*))$\rceil$])
apply(*simp add*: *d-p2r*, *rule-tac C* = λ *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* =λ *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* = λ *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*
by (*meson divide-nonneg-pos linordered-field-class.sign-simps*(*44*) *zero-less-numeral*)

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

lemma *bouncing-ball*:
PRE (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*) *FI*))*
POST (λ *s. 0* $\leq$ *s* ''x'' $\wedge$ *s* ''x'' $\leq$ *H*)
apply(*rule rel-antidomain-kleene-algebra.fbox-starI*[*of* - $\lceil\lambda$*s. 0* $\leq$ *s* ''x'' $\wedge$ *0* < *s* ''g'' $\wedge$
2 $\cdot$ *s* ''g'' $\cdot$ *s* ''x'' = *2* $\cdot$ *s* ''g'' $\cdot$ *H* $-$ (*s* ''v'' $\cdot$ *s* ''v'')$\rceil$])
apply(*simp*, *simp add*: *d-p2r*)
apply(*subgoal-tac*
rdom $\lceil\lambda$*s. 0* $\leq$ *s* ''x'' $\wedge$ *0* < *s* ''g'' $\wedge$ *2* $\cdot$ *s* ''g'' $\cdot$ *s* ''x'' = *2* $\cdot$ *s* ''g'' $\cdot$ *H* $-$ *s* ''v'' $\cdot$ *s* ''v''$\rceil$
$\subseteq$ *wp* (*ODEsystem* [(''x'', $\lambda s.$ *s* ''v''), (''v'', $\lambda s.$ $-$ *s* ''g'')] *with* ($\lambda s.$ *0* $\leq$ *s* ''x'')

)
$\lceil inf\ (sup\ (-\ (\lambda s.\ s\ ''x'' = 0))\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x''$
$=$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))$
$(sup\ (\lambda s.\ s\ ''x'' = 0)\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' =$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))\rceil)$
apply(*simp add*: *d-p2r*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = ((t_C\ \ 0) \prec (t_V\ \ ''g''))$ **and** *uInput*=$[t_V\ \ ''v'', \ominus\ t_V\ \ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''g''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ 2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ \ 2) \odot (t_V\ \ ''g'') \odot (t_C\ \ H) \oplus (\ominus\ ((t_V\ \ ''v'') \odot (t_V\ \ ''v'')))$
$\doteq (t_C\ \ 2) \odot (t_V\ \ ''g'') \odot (t_V\ \ ''x'')$ **and** *uInput*=$[t_V\ \ ''v'', \ominus\ t_V\ \ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''g''$ **in** *allE*, *simp*)
apply(*rule dWeakening*, *clarsimp*)
using *bouncing-ball-invariant* **by** *auto*

declare *d-p2r* [*simp*]

end