Verification of Hybrid Systems

Jonathan Julián Huerta y Munive

September 21, 2020

Abstract

These components formalise a semantic framework for the deductive verification of hybrid systems. They support reasoning about continuous evolutions of hybrid programs by modelling them as vector fields or continuous dynamical systems. As the framework is modular, the components can reason in the style of differential dynamic logic, Hoare logic and the weakest liberal precondition calculus. They can verify continuous evolutions directly, with invariants or by providing solutions. Laws of Kleene algebra (with tests or modal) or categorical predicate transformers complement the verification condition generation process. Extensions to Isabelle/HOL's libraries of analysis improve the verification process as evidenced by our formalisation of affine systems of ODEs. Examples show the approach at work.

Contents

0.1 Introductory Remarks

These theories exemplify a framework for creating verification components for hybrid programs, a model of hybrid systems. Using algebras for programs we obtain verification conditions for regular and while programs. For instance, using Kleene algebras with tests we obtain differential Hoare logic ($\mathsf{d}\mathcal{H}$), a minimal logic for verification of hybrid programs. Furthermore, by adding a refinement operation to this implementation we get a hybrid version of Morgan's refinement calculus [?]. However, following [?], using modal Kleene algebra that subsumes the propositional part of dynamic logic, we obtain a weakest liberal precondition calculus based on predicate transformers. In this setting, we can derive rules of differential dynamic logic [?] to reason in the style of this logic. Alternatively we also use categorical predicate transformers as formalised in [?]. The resulting verification conditions generated are entirely about the dynamics that describe the continuous evolution of the hybrid system. The dynamics are formalised with flows and vector fields for systems of ordinary differential equations (ODEs) of [?]. The components support reasoning with vector fields by annotating differential invariants or by providing the solution of the system of ODEs; otherwise, the flow is enough for verification of the continuous evolution. We formalise several rules for derivatives that, when supplied to Isabelle's *auto* method, enhance the automation of the process of discharging proof obligations.

The components also benefit from mathematical formalisations in Isabelle/HOL. As evidence we show that affine systems of ODEs satisfy the conditions for existence and uniqueness of solutions and we provide the general solution for the time-independent case. That is, if there is a matrix-valued function $A : \mathbb{R} \to M_{n\times n}(\mathbb{R})$ and vector function $B : \mathbb{R} \to \mathbb{R}^n$ such that the system of ODEs $x'\,t = f\,(t, x\,t)$ can be rewritten as $x'\,t = A\cdot(x\,t)+B\,t$, then that system is affine and satisfies Picard-Lindelöf's theorem. As a consequence, the associated linear system of ODEs is $x'\,t = A \cdot (x\,t)$ also has a unique solution. When the functions A and B are constant, we provide its general solution in terms of the matrix exponential over the Banach space of square matrices which we introduce as a new type. To simplify formalisations with this general solution, we include some results about diagonalisation and proof-automation for matrix operations.

We prove correctness specifications for several hybrid systems with our various versions of the components. In addition to these implementations, for ease of use, we also present a stand alone light-weight variant of the verification components with predicate transformers that does not depend on other AFP entries.

Background information on differential dynamic logic and some of its variants can be found in [?, ?], the general shallow embedding approach for building verification components with Isabelle can be found in [?]. For more details on modal Kleene algebra see [?]. For a technical detailed overview of the verification components in these Isabelle theories, see our work [?, ?, ?, ?]

0.2 Hybrid Systems Preliminaries

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

theory *HS-Preliminaries*
  imports *Ordinary-Differential-Equations.Picard-Lindeloef-Qualitative*
begin

— Syntax

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

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

0.2.1 Real numbers

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+*

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* **by** (*auto simp*: *cball-def ball-def dist-norm field-simps*)

lemma *is-interval-real-nonneg*[*simp*]: *is-interval* (*Collect* (($\leq$) (*0::real*)))
  by (*auto simp*: *is-interval-def*)

lemma *norm-rotate-eq*[*simp*]:
  fixes x :: $'a$:: {*banach,real-normed-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

0.2.2 Single variable derivatives

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 *has-vderiv-on-add*[*THEN has-vderiv-on-eq-rhs*, *poly-derivatives*]
    and *has-vderiv-on-diff*[*THEN has-vderiv-on-eq-rhs*, *poly-derivatives*]
    and *has-vderiv-on-mult*[*THEN has-vderiv-on-eq-rhs*, *poly-derivatives*]
    and *has-vderiv-on-ln*[*poly-derivatives*]

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

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

lemma *vderiv-npowI*[*poly-derivatives*]:

fixes f::*real* $\Rightarrow$ *real*
assumes $n \geq 1$ **and** $D\ f = f'$ *on* T **and** $g = (\lambda t.\ n * (f'\ t) * (f\ t)\hat{}(n-1))$
shows $D\ (\lambda t.\ (f\ t)\hat{}n) = g$ *on* T
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def*
by (*auto intro*: *derivative-eq-intros simp*: *field-simps*)

lemma *vderiv-divI*[*poly-derivatives*]:
assumes $\forall t \in T.\ g\ t \neq (0::real)$ **and** $D\ f = f'$ *on* T**and** $D\ g = g'$ *on* T
and $h = (\lambda t.\ (f'\ t * g\ t - f\ t * (g'\ t))\ /\ (g\ t)\hat{}2)$
shows $D\ (\lambda t.\ (f\ t)/(g\ t)) = h$ *on* T
apply(*subgoal-tac* $(\lambda t.\ (f\ t)/(g\ t)) = (\lambda t.\ (f\ t) * (1/(g\ t)))$)
apply(*erule ssubst*, *rule poly-derivatives*(5)[*OF assms*(2)])
apply(*rule vderiv-on-composeI*[**where** g=g **and** f=$\lambda t.\ 1/t$ **and** f'=$\lambda t.\ -\ 1/t\hat{}2$, *OF* - *assms*(3)])
apply(*subst has-vderiv-on-def*, *subst has-vector-derivative-def*, *clarsimp*)
using *assms*(1) **apply**(*force intro*!: *derivative-eq-intros simp*: *fun-eq-iff power2-eq-square*)
using *assms* **by** (*auto simp*: *field-simps power2-eq-square*)

lemma *vderiv-cosI*[*poly-derivatives*]:
assumes $D\ (f::real \Rightarrow real) = f'$ *on* T **and** $g = (\lambda t.\ -\ (f'\ t) * sin\ (f\ t))$
shows $D\ (\lambda t.\ cos\ (f\ t)) = g$ *on* T
apply(*rule vderiv-on-composeI*[*OF* - *assms*(1), *of* $\lambda t.\ cos\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def*
by (*auto intro*!: *derivative-eq-intros simp*: *assms*)

lemma *vderiv-sinI*[*poly-derivatives*]:
assumes $D\ (f::real \Rightarrow real) = f'$ *on* T **and** $g = (\lambda t.\ (f'\ t) * cos\ (f\ t))$
shows $D\ (\lambda t.\ sin\ (f\ t)) = g$ *on* T
apply(*rule vderiv-on-composeI*[*OF* - *assms*(1), *of* $\lambda t.\ sin\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def*
by (*auto intro*!: *derivative-eq-intros simp*: *assms*)

lemma *vderiv-expI*[*poly-derivatives*]:
assumes $D\ (f::real \Rightarrow real) = f'$ *on* T **and** $g = (\lambda t.\ (f'\ t) * exp\ (f\ t))$
shows $D\ (\lambda t.\ exp\ (f\ t)) = g$ *on* T
apply(*rule vderiv-on-composeI*[*OF* - *assms*(1), *of* $\lambda t.\ exp\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def*
by (*auto intro*!: *derivative-eq-intros simp*: *assms*)

— Examples for checking derivatives

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

lemma $a \neq 0 \Longrightarrow D\ (\lambda t.\ t/a) = (\lambda t.\ 1/a)$ *on* T
by (*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

lemma $(a::real) \neq 0 \Longrightarrow D\ f = f'$ *on* $T \Longrightarrow g = (\lambda t.\ (f'\ t)/a) \Longrightarrow D\ (\lambda t.\ (f\ t)/a) = g$ *on* T
by (*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

lemma $\forall t \in T.\ f\ t \neq (0::real) \Longrightarrow D\ f = f'$ *on* $T \Longrightarrow g = (\lambda t.\ -\ a * f'\ t\ /\ (f\ t)\hat{}2) \Longrightarrow$
$D\ (\lambda t.\ a/(f\ t)) = g$ *on* T
by (*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

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 $D\ x = x'$ *on* $(\lambda\tau.\ \tau + t)$ ` $T \Longrightarrow D\ (\lambda\tau.\ x\ (\tau + t)) = (\lambda\tau.\ x'\ (\tau + t))$ *on* T

by (*rule vderiv-on-composeI*, *auto intro*: *poly-derivatives*)

lemma $a \neq 0 \Longrightarrow D\ (\lambda t.\ t/a) = (\lambda t.\ 1/a)\ on\ T$
by (*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

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 simp*: *power2-eq-square*)

lemma $c \neq 0 \Longrightarrow D\ (\lambda t. - a3 * exp\ (t\hat{}3\ /\ c) + a1 * sin\ t + a2 * t\hat{}2) =$
$(\lambda t.\ a1 * cos\ t + 2 * a2 * t - 3 * a3 * t\hat{}2\ /\ c * exp\ (t\hat{}3\ /\ c))\ on\ T$
by(*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c))) =$
$(\lambda t. - 4 * a * t\hat{}3 * sin\ (t\hat{}4)\ /\ c * cos\ (cos\ (t\hat{}4)\ /\ c) * exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c)))\ on\ T$
by (*intro poly-derivatives*) (*auto intro*!: *poly-derivatives simp*: *power2-eq-square*)

0.2.3 Intermediate Value Theorem

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

lemma *IVT-two-functions-real-ivl*:
fixes f :: *real* $\Rightarrow$ *real*
assumes *conts*: *continuous-on* $\{a--b\}$ *f continuous-on* $\{a--b\}$ *g*
and *ahyp*: $f\ a < g\ a$ **and** *bhyp*: $g\ b < f\ b$
shows $\exists x \in \{a--b\}.\ f\ x = g\ x$
proof(*cases* $a \leq b$)
case *True*
then show *?thesis*
using *IVT-two-functions assms*
unfolding *closed-segment-eq-real-ivl* **by** *auto*
next
case *False*
hence $a \geq b$
by *auto*
hence *continuous-on* $\{b..a\}$ *f continuous-on* $\{b..a\}$ *g*
using *conts False* **unfolding** *closed-segment-eq-real-ivl* **by** *auto*
hence $\exists x \in \{b..a\}.\ g\ x = f\ x$
using *IVT-two-functions*[*of b a g f*] *assms*(*3*,*4*) *False* **by** *auto*
then show *?thesis*
using ‹$a \geq b$› **unfolding** *closed-segment-eq-real-ivl* **by** *auto force*
qed

0.2.4 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* $(\lambda x.\ \forall\, i.\ P\ i\ x)\ F$
proof(*unfold eventually-def*)
let $?F = Rep\text{-}filter\ F$
have *obs*: $\forall\, i.\ ?F\ (P\ i)$
using *h* **by** *auto*
have $?F\ (\lambda x.\ \forall\, i \in UNIV.\ P\ i\ x)$
apply(*rule finite-induct*)
by(*auto intro*: *eventually-conj simp*: *obs h*)
thus $?F\ (\lambda x.\ \forall\, i.\ P\ i\ x)$
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* $(\lambda x.\ \forall\, i.\ P\ i\ x)\ F$
using *h1 eventually-all-finite2* **by** *blast*
thus *eventually* Q F
unfolding *eventually-def*
using *h2 eventually-mono* **by** *auto*
qed

0.2.5 Multivariable derivatives

lemma *frechet-vec-lambda*:
fixes $f{::}real \Rightarrow ('a{::}banach)\hat{}('m{::}finite)$ **and** $x{::}real$ **and** $T{::}real\ set$
defines $x_0 \equiv netlimit\ (at\ x\ within\ T)$ **and** $m \equiv real\ CARD('m)$
assumes $\forall\, i.\ ((\lambda y.\ (f\ y\ \$\ i - f\ x_0\ \$\ i - (y - x_0) *_R f'\ x\ \$\ i) /_R (\|y - x_0\|)) \longrightarrow 0)\ (at\ x\ within\ T)$
shows $((\lambda y.\ (f\ y - f\ x_0 - (y - x_0) *_R f'\ x) /_R (\|y - x_0\|)) \longrightarrow 0)\ (at\ x\ within\ T)$
proof(*simp add*: *tendsto-iff* , *clarify*)
fix $\varepsilon{::}real$ **assume** $0 < \varepsilon$
let $?\Delta = \lambda y.\ y - x_0$ **and** $?\Delta f = \lambda y.\ f\ y - f\ x_0$

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

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

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

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

lemma *has-derivative-vec-lambda*:
fixes $f::real \Rightarrow ('a::banach)\,\hat{}\,('n::finite)$
assumes $\forall i.\ D\ (\lambda t.\ f\,t\ \$\ i) \mapsto (\lambda\ h.\ h *_R f'\ x\ \$\ i)\ (at\ x\ within\ T)$

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

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

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

end

0.3 Ordinary Differential Equations

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

theory *HS-ODEs*
imports *HS-Preliminaries*
begin

0.3.1 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* :: $(real \Rightarrow 'a \Rightarrow ('a::real\text{-}normed\text{-}vector)) \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a \Rightarrow (real \Rightarrow 'a)\ set\ (Sols)$
where $Sols\ f\ U\ S\ t_0\ s = \{X \in U\ s \to S.\ (D\ X = (\lambda t.\ f\ t\ (X\ t))\ on\ U\ s) \wedge X\ t_0 = s \wedge t_0 \in U\ s\}$

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

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

lemma *in-ivp-sols-subset*:
$t_0 \in (U\ s) \Longrightarrow (U\ s) \subseteq (T\ s) \Longrightarrow X \in Sols\ f\ T\ S\ t_0\ s \Longrightarrow X \in Sols\ f\ U\ S\ t_0\ s$

apply(*rule ivp-solsI*)
using *ivp-solsD(1,2) has-vderiv-on-subset* **apply** *blast+*
by (*drule ivp-solsD(3)*) *auto*

abbreviation $down\ U\ t \equiv \{\tau \in U.\ \tau \le t\}$

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

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

definition *g-orbital* :: $(real \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow real \Rightarrow$
$('a{::}real\text{-}normed\text{-}vector) \Rightarrow 'a\ set$
where $g\text{-}orbital\ f\ G\ U\ S\ t_0\ s = \bigcup\{\gamma\ X\ G\ (U\ s)\ |X.\ X \in ivp\text{-}sols\ f\ U\ S\ t_0\ s\}$

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

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

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

lemma $g\text{-}orbital\ f\ G\ U\ S\ t_0\ s = \{X\ t\ |t\ X.\ X\ t \in \gamma\ X\ G\ (U\ s) \wedge X \in Sols\ f\ U\ S\ t_0\ s\}$
unfolding *g-orbital-eq g-orbit-eq* **by** *auto*

lemma $X \in Sols\ f\ U\ S\ t_0\ s \Longrightarrow \gamma\ X\ G\ (U\ s) \subseteq g\text{-}orbital\ f\ G\ U\ S\ t_0\ s$
unfolding *g-orbital-eq g-orbit-eq* **by** *auto*

lemma $g\text{-}orbital\ f\ G\ U\ S\ t_0\ s \subseteq g\text{-}orbital\ f\ (\lambda s.\ True)\ U\ S\ t_0\ s$
unfolding *g-orbital-eq* **by** *auto*

no-notation *g-orbit* (γ)

0.3.2 Differential Invariants

definition *diff-invariant* :: $('a \Rightarrow bool) \Rightarrow (real \Rightarrow ('a{::}real\text{-}normed\text{-}vector) \Rightarrow 'a) \Rightarrow$
$('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow real \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$
where $diff\text{-}invariant\ I\ f\ U\ S\ t_0\ G \equiv (\bigcup \circ (\mathcal{P}\ (g\text{-}orbital\ f\ G\ U\ S\ t_0)))\ \{s.\ I\ s\} \subseteq \{s.\ I\ s\}$

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

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

lemma $diff\text{-}invariant\ I\ f\ U\ S\ t_0\ (\lambda s.\ True) \Longrightarrow diff\text{-}invariant\ I\ f\ U\ S\ t_0\ G$

unfolding *diff-invariant-eq* **by** *auto*

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

lemma *diff-invariant-eq-rule* [*diff-invariant-rules*]:
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow$ *is-interval* $(U\ s)$
and dX: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (D\ (\lambda\tau.\ \mu(X\ \tau){-}\nu(X\ \tau)) = ((*_R)\ 0)\ on\ U(X\ t_0))$
shows *diff-invariant* $(\lambda s.\ \mu\ s = \nu\ s)\ f\ U\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix $X\ t$
assume *xivp*:$D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U\ (X\ t_0)\ \mu\ (X\ t_0) = \nu\ (X\ t_0)\ X \in U\ (X\ t_0) \to S$
and *tHyp*:$t \in U\ (X\ t_0)$ **and** *t0Hyp*: $t_0 \in U\ (X\ t_0)$
hence $\{t_0{-}{-}t\} \subseteq U\ (X\ t_0)$
using *closed-segment-subset-interval*[*OF Uhyp t0Hyp tHyp*] **by** *blast*
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \tau *_R 0)\ on\ \{t_0{-}{-}t\}$
using *has-vderiv-on-subset*[*OF dX*[*OF xivp*(*1*)]] **by** *auto*
then obtain τ **where** $\mu\ (X\ t) - \nu\ (X\ t) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (t - t_0) * \tau *_R 0$
using *mvt-very-simple-closed-segmentE* **by** *blast*
thus $\mu\ (X\ t) = \nu\ (X\ t)$
by (*simp add*: *xivp*(*2*))
qed

lemma *diff-invariant-leq-rule* [*diff-invariant-rules*]:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow$ *is-interval* $(U\ s)$
and *Gg*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall\tau{\in}U(X\ t_0).\ \tau > t_0 \longrightarrow G\ (X\ \tau) \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau))$
and *Gl*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall\tau{\in}U(X\ t_0).\ \tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))$
and dX: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow D\ (\lambda\tau.\ \mu(X\ \tau){-}\nu(X\ \tau)) = (\lambda\tau.\ \mu'(X\ \tau){-}\nu'(X\ \tau))\ on\ U(X\ t_0)$
shows *diff-invariant* $(\lambda s.\ \nu\ s \leq \mu\ s)\ f\ U\ S\ t_0\ G$
proof(*simp-all add*: *diff-invariant-eq ivp-sols-def*, *safe*)
fix $X\ t$ **assume** *Ghyp*: $\forall\tau.\ \tau \in U\ (X\ t_0) \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
assume *xivp*: $D\ X = (\lambda x.\ f\ x\ (X\ x))\ on\ U\ (X\ t_0)\ \nu\ (X\ t_0) \leq \mu\ (X\ t_0)\ X \in U\ (X\ t_0) \to S$
assume *tHyp*: $t \in U\ (X\ t_0)$ **and** *t0Hyp*: $t_0 \in U\ (X\ t_0)$
hence *obs1*: $\{t_0{-}{-}t\} \subseteq U\ (X\ t_0)\ \{t_0{<}{-}{-}{<}t\} \subseteq U\ (X\ t_0)$
using *closed-segment-subset-interval*[*OF Uhyp t0Hyp tHyp*] *xivp*(*3*) *segment-open-subset-closed*
by (*force*, *metis PiE* ‹$X\ t_0 \in S \Longrightarrow \{t_0{-}{-}t\} \subseteq U\ (X\ t_0)$› *dual-order.trans*)
hence *obs2*: $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ \{t_0{-}{-}t\}$
using *has-vderiv-on-subset*[*OF dX*[*OF xivp*(*1*)]] **by** *auto*
{**assume** $t \neq t_0$
then obtain r **where** *rHyp*: $r \in \{t_0{<}{-}{-}{<}t\}$
and $(\mu(X\ t){-}\nu(X\ t)) - (\mu(X\ t_0){-}\nu(X\ t_0)) = (\lambda\tau.\ \tau{*}(\mu'(X\ r){-}\nu'(X\ r)))\ (t - t_0)$
using *mvt-simple-closed-segmentE obs2* **by** *blast*
hence *mvt*: $\mu(X\ t){-}\nu(X\ t) = (t - t_0){*}(\mu'(X\ r){-}\nu'(X\ r)) + (\mu(X\ t_0){-}\nu(X\ t_0))$
by *force*
have *primed*: $\bigwedge\tau.\ \tau \in U\ (X\ t_0) \Longrightarrow \tau > t_0 \Longrightarrow G\ (X\ \tau) \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in U\ (X\ t_0) \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *Gg*[*OF xivp*(*1*)] *Gl*[*OF xivp*(*1*)] **by** *auto*
have $t > t_0 \Longrightarrow r > t_0 \wedge G\ (X\ r)\ \neg\ t_0 \leq t \Longrightarrow r < t_0\ r \in U\ (X\ t_0)$
using ‹$r \in \{t_0{<}{-}{-}{<}t\}$› *obs1 Ghyp*
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl* **by** *auto*
moreover have $r > t_0 \Longrightarrow G\ (X\ r) \Longrightarrow (\mu'(X\ r){-}\ \nu'(X\ r)) \geq 0\ r < t_0 \Longrightarrow (\mu'(X\ r){-}\nu'(X\ r)) \leq 0$
using *primed*(*1*,*2*)[*OF* ‹$r \in U\ (X\ t_0)$›] **by** *auto*
ultimately have $(t - t_0) * (\mu'(X\ r){-}\nu'(X\ r)) \geq 0$
by (*case-tac* $t \geq t_0$, *force*, *auto simp*: *split-mult-pos-le*)
hence $(t - t_0) * (\mu'(X\ r){-}\nu'(X\ r)) + (\mu(X\ t_0){-}\nu(X\ t_0)) \geq 0$
using *xivp*(*2*) **by** *auto*

hence $\nu\ (X\ t) \leq \mu\ (X\ t)$
using *mvt* **by** *simp*}
thus $\nu\ (X\ t) \leq \mu\ (X\ t)$
using *xivp* **by** *blast*
qed

lemma *diff-invariant-less-rule* [*diff-invariant-rules*]:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow$ *is-interval* $(U\ s)$
and *Gg*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall\tau{\in}U(X\ t_0).\ \tau > t_0 \longrightarrow G\ (X\ \tau) \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau))$
and *Gl*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall\tau{\in}U(X\ t_0).\ \tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))$
and *dX*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow D\ (\lambda\tau.\ \mu(X\ \tau) - \nu(X\ \tau)) = (\lambda\tau.\ \mu'(X\ \tau) - \nu'(X\ \tau))\ on\ U(X\ t_0)$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
proof(*simp-all add*: *diff-invariant-eq ivp-sols-def*, *safe*)
fix $X\ t$ **assume** *Ghyp*: $\forall\tau.\ \tau \in U\ (X\ t_0) \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
assume *xivp*: $D\ X = (\lambda x.\ f\,x\ (X\ x))\ on\ U\ (X\ t_0)\ \nu\ (X\ t_0) < \mu\ (X\ t_0)\ X \in U\ (X\ t_0) \rightarrow S$
assume *tHyp*: $t \in U\ (X\ t_0)$ **and** *t0Hyp*: $t_0 \in U\ (X\ t_0)$
hence *obs1*: $\{t_0{-}{-}t\} \subseteq U\ (X\ t_0)\ \{t_0{<}{-}{-}{<}t\} \subseteq U\ (X\ t_0)$
using *closed-segment-subset-interval*[*OF Uhyp t0Hyp tHyp*] *xivp*(*3*) *segment-open-subset-closed*
by (*force*, *metis PiE* ‹$X\ t_0 \in S \Longrightarrow \{t_0{-}{-}t\} \subseteq U\ (X\ t_0)$› *dual-order.trans*)
hence *obs2*: $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ \{t_0{-}{-}t\}$
using *has-vderiv-on-subset*[*OF dX*[*OF xivp*(*1*)]] **by** *auto*
{**assume** $t \neq t_0$
then obtain r **where** *rHyp*: $r \in \{t_0{<}{-}{-}{<}t\}$
and $(\mu(X\ t) - \nu(X\ t)) - (\mu(X\ t_0) - \nu(X\ t_0)) = (\lambda\tau.\ \tau{*}(\mu'(X\ r) - \nu'(X\ r)))\ (t - t_0)$
using *mvt-simple-closed-segmentE obs2* **by** *blast*
hence *mvt*: $\mu(X\ t) - \nu(X\ t) = (t - t_0){*}(\mu'(X\ r) - \nu'(X\ r)) + (\mu(X\ t_0) - \nu(X\ t_0))$
by *force*
have *primed*: $\bigwedge\tau.\ \tau \in U\ (X\ t_0) \Longrightarrow \tau > t_0 \Longrightarrow G\ (X\ \tau) \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in U\ (X\ t_0) \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *Gg*[*OF xivp*(*1*)] *Gl*[*OF xivp*(*1*)] **by** *auto*
have $t > t_0 \Longrightarrow r > t_0 \wedge G\ (X\ r)\ \neg\ t_0 \leq t \Longrightarrow r < t_0\ r \in U\ (X\ t_0)$
using ‹$r \in \{t_0{<}{-}{-}{<}t\}$› *obs1 Ghyp*
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl* **by** *auto*
moreover have $r > t_0 \Longrightarrow G\ (X\ r) \Longrightarrow (\mu'(X\ r) - \nu'(X\ r)) \geq 0\ r < t_0 \Longrightarrow (\mu'(X\ r) - \nu'(X\ r)) \leq 0$
using *primed*(*1*,*2*)[*OF* ‹$r \in U\ (X\ t_0)$›] **by** *auto*
ultimately have $(t - t_0) * (\mu'(X\ r) - \nu'(X\ r)) \geq 0$
by (*case-tac* $t \geq t_0$, *force*, *auto simp*: *split-mult-pos-le*)
hence $(t - t_0) * (\mu'(X\ r) - \nu'(X\ r)) + (\mu(X\ t_0) - \nu(X\ t_0)) > 0$
using *xivp*(*2*) **by** *auto*
hence $\nu\ (X\ t) < \mu\ (X\ t)$
using *mvt* **by** *simp*}
thus $\nu\ (X\ t) < \mu\ (X\ t)$
using *xivp* **by** *blast*
qed

lemma *diff-invariant-nleq-rule*:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
shows *diff-invariant* $(\lambda s.\ \neg\ \nu\ s \leq \mu\ s)\ f\ U\ S\ t_0\ G \longleftrightarrow$ *diff-invariant* $(\lambda s.\ \nu\ s > \mu\ s)\ f\ U\ S\ t_0\ G$
unfolding *diff-invariant-eq* **apply** *safe*
by (*clarsimp*, *erule-tac x=s* **in** *allE*, *simp*, *erule-tac x=X* **in** *ballE*, *force*, *force*)+

lemma *diff-invariant-neq-rule* [*diff-invariant-rules*]:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
assumes *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
and *diff-invariant* $(\lambda s.\ \nu\ s > \mu\ s)\ f\ U\ S\ t_0\ G$
shows *diff-invariant* $(\lambda s.\ \nu\ s \neq \mu\ s)\ f\ U\ S\ t_0\ G$
proof(*unfold diff-invariant-eq*, *clarsimp*)

fix $s{::}'a$ **and** $X{::}real \Rightarrow 'a$ **and** $t{::}real$
assume $\nu\ s \neq \mu\ s$ **and** *Xhyp*: $X \in Sols\ f\ U\ S\ t_0\ s$
and *thyp*: $t \in U\ s$ **and** *Ghyp*: $\forall \tau.\ \tau \in U\ s \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
hence $\nu\ s < \mu\ s \vee \nu\ s > \mu\ s$
by *linarith*
moreover have $\nu\ s < \mu\ s \Longrightarrow \nu\ (X\ t) < \mu\ (X\ t)$
using *assms(1) Xhyp thyp Ghyp* **unfolding** *diff-invariant-eq* **by** *auto*
moreover have $\nu\ s > \mu\ s \Longrightarrow \nu\ (X\ t) > \mu\ (X\ t)$
using *assms(2) Xhyp thyp Ghyp* **unfolding** *diff-invariant-eq* **by** *auto*
ultimately show $\nu\ (X\ t) = \mu\ (X\ t) \Longrightarrow False$
by *auto*
qed

lemma *diff-invariant-neq-rule-converse*:
fixes $\mu{::}'a{::}banach \Rightarrow real$
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow is\text{-}interval\ (U\ s)\ \bigwedge s\ t.\ s \in S \Longrightarrow t \in U\ s \Longrightarrow t_0 \leq t$
and *conts*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow continuous\text{-}on\ (\mathcal{P}\ X\ (U\ (X\ t_0)))\ \nu$
$\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow continuous\text{-}on\ (\mathcal{P}\ X\ (U\ (X\ t_0)))\ \mu$
and *dI*:*diff-invariant* $(\lambda s.\ \nu\ s \neq \mu\ s)\ f\ U\ S\ t_0\ G$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
proof(*unfold diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix $X\ t$ **assume** *Ghyp*: $\forall \tau.\ \tau \in U\ (X\ t_0) \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
assume *xivp*: $D\ X = (\lambda x.\ f\ x\ (X\ x))\ on\ U\ (X\ t_0)\ \nu\ (X\ t_0) < \mu\ (X\ t_0)\ X \in U\ (X\ t_0) \rightarrow S$
assume *tHyp*: $t \in U\ (X\ t_0)$ **and** *t0Hyp*: $t_0 \in U\ (X\ t_0)$
hence $t_0 \leq t$ **and** $\mu\ (X\ t) \neq \nu\ (X\ t)$
using *xivp(3) Uhyp(2)* **apply** *force*
using *dI tHyp xivp(2) Ghyp ivp-solsI*[*of* $X\ f\ U\ X\ t_0$, *OF xivp(1) - xivp(3) t0Hyp*]
unfolding *diff-invariant-eq* **by** *force*
moreover
{**assume** *ineq2*:$\nu\ (X\ t) > \mu\ (X\ t)$
note *continuous-on-compose*[*OF vderiv-on-continuous-on*[*OF xivp(1)*]]
hence *continuous-on* $(U\ (X\ t_0))\ (\nu \circ X)$ **and** *continuous-on* $(U\ (X\ t_0))\ (\mu \circ X)$
using *xivp(1) conts* **by** *blast+*
also have $\{t_0{-}{-}t\} \subseteq U\ (X\ t_0)$
using *closed-segment-subset-interval*[*OF Uhyp(1) t0Hyp tHyp*] *xivp(3) t0Hyp* **by** *auto*
ultimately have *continuous-on* $\{t_0{-}{-}t\}\ (\lambda\tau.\ \nu\ (X\ \tau))$
and *continuous-on* $\{t_0{-}{-}t\}\ (\lambda\tau.\ \mu\ (X\ \tau))$
using *continuous-on-subset* **by** *auto*
then obtain τ **where** $\tau \in \{t_0{-}{-}t\}\ \mu\ (X\ \tau) = \nu\ (X\ \tau)$
using *IVT-two-functions-real-ivl*[*OF - - xivp(2) ineq2*] **by** *force*
hence $\forall r{\in}down\ (U\ (X\ t_0))\ \tau.\ G\ (X\ r)$ **and** $\tau \in U\ (X\ t_0)$
using *Ghyp* $\langle\tau \in \{t_0{-}{-}t\}\rangle\ \langle t_0 \leq t\rangle\ \langle\{t_0{-}{-}t\} \subseteq U\ (X\ t_0)\rangle$
by (*auto simp*: *closed-segment-eq-real-ivl*)
hence $\mu\ (X\ \tau) \neq \nu\ (X\ \tau)$
using *dI tHyp xivp(2) ivp-solsI*[*of* $X\ f\ U\ X\ t_0$, *OF xivp(1) - xivp(3) t0Hyp*]
unfolding *diff-invariant-eq* **by** *force*
hence *False*
using $\langle\mu\ (X\ \tau) = \nu\ (X\ \tau)\rangle$ **by** *blast*}
ultimately show $\nu\ (X\ t) < \mu\ (X\ t)$
by *fastforce*
qed

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

lemma *diff-invariant-disj-rule* [*diff-invariant-rules*]:
assumes *diff-invariant* $I_1\ f\ U\ S\ t_0\ G$

and *diff-invariant* I_2 *f U S* t_0 *G*
shows *diff-invariant* ($\lambda s.$ I_1 *s* $\vee$ I_2 *s*) *f U S* t_0 *G*
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

0.3.3 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*)

lemma *ll-on-open*: *ll-on-open T f S*
using *local.general.ll-on-open-axioms* **.**

lemmas *subintervalI* = *closed-segment-subset-domain*
and *init-time-ex-ivl* = *existence-ivl-initial-time*[*OF init-time*]
and *flow-at-init*[*simp*] = *general.flow-initial-time*[*OF init-time*]

abbreviation *ex-ivl s* ≡ *existence-ivl* t_0 *s*

lemma *flow-has-vderiv-on-ex-ivl*:
assumes $s \in S$
shows *D flow* t_0 *s* = ($\lambda t.$ *f t* (*flow* t_0 *s t*)) *on ex-ivl s*
using *flow-usolves-ode*[*OF init-time* ‹$s \in S$›]
unfolding *usolves-ode-from-def solves-ode-def* **by** *blast*

lemma *flow-funcset-ex-ivl*:
assumes $s \in S$
shows *flow* t_0 *s* ∈ *ex-ivl s* → *S*
using *flow-usolves-ode*[*OF init-time* ‹$s \in S$›]
unfolding *usolves-ode-from-def solves-ode-def* **by** *blast*

lemma *flow-in-ivp-sols-ex-ivl*:
assumes $s \in S$
shows *flow* t_0 *s* ∈ *Sols f* ($\lambda s.$ *ex-ivl s*) *S* t_0 *s*
using *flow-has-vderiv-on-ex-ivl*[*OF assms*] **apply**(*rule ivp-solsI*)
apply(*simp-all add*: *init-time assms*)
by (*rule flow-funcset-ex-ivl*[*OF assms*])

lemma *csols-eq*: *csols* t_0 *s* = {(*x*, *t*). $t \in T \wedge$ *x* ∈ *Sols f* ($\lambda s.$ $\{t_0{-}{-}t\}$) *S* t_0 *s*}
unfolding *ivp-sols-def csols-def solves-ode-def*
using *closed-segment-subset-domain init-time* **by** *auto*

lemma *subset-ex-ivlI*:
Y_1 ∈ *Sols f* ($\lambda s.$ *T*) *S* t_0 *s* ⟹ $\{t_0{-}{-}t\} \subseteq T$ ⟹ $A \subseteq \{t_0{-}{-}t\}$ ⟹ *A* ⊆ *ex-ivl s*
apply(*clarsimp simp*: *existence-ivl-def*)
apply(*subgoal-tac* $t_0 \in T$, *clarsimp simp*: *csols-eq*)
apply(*rule-tac x*=Y_1 **in** *exI*, *rule-tac x*=*t* **in** *exI*, *safe*, *force*)
by (*rule in-ivp-sols-subset*[**where** *T*=$\lambda s.$ *T*], *auto*)

lemma *unique-solution*: — proved for a subset of T for general applications
  assumes $s \in S$ **and** $t_0 \in U$ **and** $t \in U$
    and *is-interval* U **and** $U \subseteq$ *ex-ivl* s
    and *xivp*: $D\ Y_1 = (\lambda t.\ f\ t\ (Y_1\ t))$ *on* $U\ Y_1\ t_0 = s\ Y_1 \in U \to S$
    and *yivp*: $D\ Y_2 = (\lambda t.\ f\ t\ (Y_2\ t))$ *on* $U\ Y_2\ t_0 = s\ Y_2 \in U \to S$
  shows $Y_1\ t = Y_2\ t$
proof−
  have $t_0 \in T$
    using *assms existence-ivl-subset* **by** *auto*
  have *key*: (*flow* t_0 s *usolves-ode* *f from* t_0) (*ex-ivl* s) S
    using *flow-usolves-ode*[*OF* ‹$t_0 \in T$› ‹$s \in S$›] **.**
  hence $\forall t{\in}U.\ Y_1\ t =$ *flow* $t_0\ s\ t$
    unfolding *usolves-ode-from-def solves-ode-def* **apply** *safe*
    by (*erule-tac* $x{=}Y_1$ **in** *allE*, *erule-tac* $x{=}U$ **in** *allE*, *auto simp*: *assms*)
  also have $\forall t{\in}U.\ Y_2\ t =$ *flow* $t_0\ s\ t$
    using *key* **unfolding** *usolves-ode-from-def solves-ode-def* **apply** *safe*
    by (*erule-tac* $x{=}Y_2$ **in** *allE*, *erule-tac* $x{=}U$ **in** *allE*, *auto simp*: *assms*)
  ultimately show $Y_1\ t = Y_2\ t$
    using *assms* **by** *auto*
qed

Applications of lemma *unique-solution*:

lemma *unique-solution-closed-ivl*:
  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 $X\ t = Y\ t$
  apply(*rule unique-solution*[*OF* ‹$s \in S$›, *of* $\{t_0{-}{-}t\}$], *simp-all add*: *assms*)
  apply(*unfold existence-ivl-def csols-eq ivp-sols-def*, *clarsimp*)
  using *xivp* ‹$t \in T$› **by** *blast*

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* $t_0\ s\ t$
  apply(*rule unique-solution*[*OF* ‹$s \in S$› *init-time-ex-ivl* ‹$t \in$ *ex-ivl* s›])
  using *flow-has-vderiv-on-ex-ivl flow-funcset-ex-ivl* ‹$s \in S$› **by** (*auto simp*: *assms*)

lemma *ivp-unique-solution*:
  assumes $s \in S$ **and** *ivl*: *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $t \in U\ s$
    and *ivp1*: $Y_1 \in$ *Sols* $f\ U\ S\ t_0\ s$ **and** *ivp2*: $Y_2 \in$ *Sols* $f\ U\ S\ t_0\ s$
  shows $Y_1\ t = Y_2\ t$
proof(*rule unique-solution*[*OF* ‹$s \in S$›, *of* $\{t_0{-}{-}t\}$], *simp-all*)
  have $t_0 \in U\ s$
    using *ivp-solsD*[*OF ivp1*] **by** *auto*
  hence *obs0*: $\{t_0{-}{-}t\} \subseteq U\ s$
    using *closed-segment-subset-interval*[*OF ivl*] ‹$t \in U\ s$› **by** *blast*
  moreover have *obs1*: $Y_1 \in$ *Sols* $f\ (\lambda s.\ \{t_0{-}{-}t\})\ S\ t_0\ s$
    by (*rule in-ivp-sols-subset*[*OF - calculation*(*1*) *ivp1*], *simp*)
  moreover have *obs2*: $Y_2 \in$ *Sols* $f\ (\lambda s.\ \{t_0{-}{-}t\})\ S\ t_0\ s$
    by (*rule in-ivp-sols-subset*[*OF - calculation*(*1*) *ivp2*], *simp*)
  ultimately show $\{t_0{-}{-}t\} \subseteq$ *ex-ivl* s
    apply(*unfold existence-ivl-def csols-eq*, *clarsimp*)
    apply(*rule-tac* $x{=}Y_1$ **in** *exI*, *rule-tac* $x{=}t$ **in** *exI*)
    using ‹$t \in U\ s$› **and** ‹$U\ s \subseteq T$› **by** *force*
  show $D\ Y_1 = (\lambda t.\ f\ t\ (Y_1\ t))$ *on* $\{t_0{-}{-}t\}$
    by (*rule ivp-solsD*[*OF in-ivp-sols-subset*[*OF - - ivp1*]], *simp-all add*: *obs0*)
  show $D\ Y_2 = (\lambda t.\ f\ t\ (Y_2\ t))$ *on* $\{t_0{-}{-}t\}$
    by (*rule ivp-solsD*[*OF in-ivp-sols-subset*[*OF - - ivp2*]], *simp-all add*: *obs0*)
  show $Y_1\ t_0 = s$ **and** $Y_2\ t_0 = s$
    using *ivp-solsD*[*OF ivp1*] *ivp-solsD*[*OF ivp2*] **by** *auto*

show $Y_1 \in \{t_0{-}{-}t\} \to S$ **and** $Y_2 \in \{t_0{-}{-}t\} \to S$
  using *ivp-solsD*[*OF obs1*] *ivp-solsD*[*OF obs2*] **by** *auto*
qed

lemma *g-orbital-orbit*:
  assumes $s \in S$ **and** *ivl*: *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$
    and *ivp*: $Y \in$ *Sols* $f\ U\ S\ t_0\ s$
  shows *g-orbital* $f\ G\ U\ S\ t_0\ s =$ *g-orbit* $Y\ G\ (U\ s)$
proof−
  have *eq1*: $\forall Z \in$ *Sols* $f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ Z\ t = Y\ t$
    by (*clarsimp*, *rule ivp-unique-solution*[*OF assms*(*1*,*2*,*3*) - - *ivp*], *auto*)
  have *g-orbital* $f\ G\ U\ S\ t_0\ s \subseteq$ *g-orbit* $(\lambda t.\ Y\ t)\ G\ (U\ s)$
  proof
    fix x **assume** $x \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
    then obtain Z **and** t
      where *z-def*: $x = Z\ t \wedge t \in U\ s \wedge (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (Z\ \tau)) \wedge Z \in$ *Sols* $f\ U\ S\ t_0\ s$
      unfolding *g-orbital-eq* **by** *auto*
    hence $\{t_0{-}{-}t\} \subseteq U\ s$
      using *closed-segment-subset-interval*[*OF ivl ivp-solsD*(*4*)[*OF ivp*]] **by** *blast*
    hence $\forall \tau{\in}\{t_0{-}{-}t\}.\ Z\ \tau = Y\ \tau$
      using *z-def* **apply** *clarsimp*
      by (*rule ivp-unique-solution*[*OF assms*(*1*,*2*,*3*) - - *ivp*], *auto*)
    thus $x \in$ *g-orbit* $Y\ G\ (U\ s)$
      using *z-def eq1* **unfolding** *g-orbit-eq* **by** *simp metis*
  qed
  moreover have *g-orbit* $Y\ G\ (U\ s) \subseteq$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
    apply(*unfold g-orbital-eq g-orbit-eq ivp-sols-def*, *clarsimp*)
    apply(*rule-tac* $x{=}t$ **in** *exI*, *rule-tac* $x{=}Y$ **in** *exI*)
    using *ivp-solsD*[*OF ivp*] **by** *auto*
  ultimately show *?thesis*
    by *blast*
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-lindeloef f1 T S* $t_0 \Longrightarrow$ *picard-lindeloef f2 T S* $t_0 \Longrightarrow$
picard-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-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*)

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

lemma *in-ivp-sols-ivl*:
assumes $t \in T$ $s \in S$
shows $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ (\lambda s.\ \{0--t\})\ S\ 0\ s$
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-closed-ivl*[*OF xivp* ‹$t \in T$›])
using ‹$s \in S$› *ivp indom* **by** *auto*

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

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

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

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

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

using *subintervalI*[*OF init-time*] **by** *auto*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{0--r/2\})$
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{0--(-r/2)\})$
using *ivp*(*1*)[*OF* - ‹$s \in S$›] **by** *auto*
also have $\{0--r/2\} = \{0--r/2\} \cup \mathit{closure}\ \{0--r/2\} \cap \mathit{closure}\ \{0--(-r/2)\}$
$\{0--(-r/2)\} = \{0--(-r/2)\} \cup \mathit{closure}\ \{0--r/2\} \cap \mathit{closure}\ \{0--(-r/2)\}$
unfolding *closed-segment-eq-real-ivl* ‹$r/2 > 0$› **by** *auto*
ultimately have *vderivs*:
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{0--r/2\} \cup \mathit{closure}\ \{0--r/2\} \cap \mathit{closure}\ \{0--(-r/2)\})$
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{0--(-r/2)\} \cup \mathit{closure}\ \{0--r/2\} \cap \mathit{closure}\ \{0--(-r/2)\})$
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 $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{-r/2--r/2\})$
using *has-vderiv-on-union*[*OF vderivs*] **by** *simp*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ \{-r/2<--<r/2\})$
using *has-vderiv-on-subset*[*OF* - *segment-open-subset-closed*[*of* $-r/2\ r/2$]] **by** *auto*
hence $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ 0\ s))\ (\mathit{at}\ 0\ \mathit{within}\ \{-r/2<--<r/2\})$
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 $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ \mathit{at}\ t\ \mathit{within}\ B$
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 $(\lambda t.\ \varphi\ t\ s) \in T \to S$
using *ivp*(*3*)[*OF* - *assms*] **by** *blast*

lemma *has-vderiv-on-domain*:
assumes $s \in S$
shows $D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ \mathit{on}\ T$
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*: $D\ (\lambda t.\ \varphi\ t\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ \mathit{at}\ t\ \mathit{within}\ B$
using *assms has-derivative-on-open*[*OF* ‹$t \in T$›] **by** *blast*
hence $t \in \mathit{interior}\ B$
using *interior-eq* **by** *auto*
thus $D\ (\lambda t.\ \varphi\ t\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ \mathit{at}\ t\ \mathit{within}\ T$
using *has-derivative-at-within-mono*[*OF* - ‹$B \subseteq T$› *Dhyp*] **by** *blast*
qed

lemma *in-ivp-sols*:
assumes $s \in S$ **and** $0 \in U\ s$ **and** $U\ s \subseteq T$
shows $(\lambda t.\ \varphi\ t\ s) \in \mathit{Sols}\ (\lambda t.\ f)\ U\ S\ 0\ s$
apply(*rule in-ivp-sols-subset*[*OF* - - *ivp-solsI*, *of* - - - $\lambda s.\ T$])

using *ivp*(*2*)[*OF* ‹$s \in S$›] *has-vderiv-on-domain*[*OF* ‹$s \in S$›]
in-domain[*OF* ‹$s \in S$›] *assms* **by** *auto*

lemma *eq-solution*:
assumes $s \in S$ **and** *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $t \in U\ s$
and *xivp*: $X \in Sols\ (\lambda t.\ f)\ U\ S\ 0\ s$
shows $X\ t = \varphi\ t\ s$
apply(*rule ivp-unique-solution*[*OF assms*], *rule in-ivp-sols*)
by (*simp-all add*: *ivp-solsD*(*4*)[*OF xivp*] *assms*)

lemma *ivp-sols-collapse*:
assumes $T = UNIV$ **and** $s \in S$
shows $Sols\ (\lambda t.\ f)\ (\lambda s.\ T)\ S\ 0\ s = \{(\lambda t.\ \varphi\ t\ s)\}$
apply (*safe*, *simp-all add*: *fun-eq-iff*, *clarsimp*)
apply(*rule eq-solution*[*of* - $\lambda s.\ T$]; *simp add*: *assms*)
by (*rule in-ivp-sols*; *simp add*: *assms*)

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

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

lemma *g-orbital-collapses*:
assumes $s \in S$ **and** *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $0 \in U\ s$
shows *g-orbital* $(\lambda t.\ f)\ G\ U\ S\ 0\ s = \{\varphi\ t\ s|\ t.\ t \in U\ s \wedge (\forall\tau\in down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s))\}$
apply (*subst g-orbital-orbit*[*of* - - $\lambda t.\ \varphi\ t\ s$], *simp-all add*: *assms g-orbit-eq*)
by (*rule in-ivp-sols*, *simp-all add*: *assms*)

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

lemma *orbit-eq*:
assumes $s \in S$
shows $\gamma^\varphi\ s = \{\varphi\ t\ s|\ t.\ t \in T\}$
apply(*unfold orbit-def*, *subst g-orbital-collapses*)
by (*simp-all add*: *assms init-time interval-time*)

lemma *true-g-orbit-eq*:
assumes $s \in S$
shows *g-orbit* $(\lambda t.\ \varphi\ t\ s)\ (\lambda s.\ True)\ T = \gamma^\varphi\ s$
unfolding *g-orbit-eq orbit-eq*[*OF assms*] **by** *simp*

end

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

end

0.4 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-ODEs*

begin

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

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

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

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

0.4.1 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-anti*: $\forall s.\ F_1\ s \subseteq F_2\ s \Longrightarrow |F_2]\ P \leq |F_1]\ P$
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 starting with *skip*.

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

Next, we introduce assignments and their wlps.

definition *vec-upd* :: *'a^'n* $\Rightarrow$ *'n* $\Rightarrow$ *'a* $\Rightarrow$ *'a^'n*
where *vec-upd s i a* = (χ *j*. ((($) *s*)(*i* := *a*)) *j*)

lemma *vec-upd-eq*: *vec-upd s i a* = (χ *j*. *if j* = *i then a else s$j*)
by (*simp add*: *vec-upd-def*)

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

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

definition *nondet-assign* :: $'n \Rightarrow 'a\hat{\ }'n \Rightarrow ('a\hat{\ }'n)$ *set* ((*2*- ::= *?*) [*70*] *61*)
where $(x ::= ?) = (\lambda s.\ \{(vec\text{-}upd\ s\ x\ k)|k.\ True\})$

lemma *fbox-nondet-assign*[*simp*]: $|x ::= ?]\ P = (\lambda s.\ \forall k.\ P\ (\chi\ j.\ if\ j = x\ then\ k\ else\ s\$j))$
unfolding *fbox-def nondet-assign-def vec-upd-eq* **apply**(*simp add*: *fun-eq-iff*, *safe*)
by (*erule-tac* $x{=}(\chi\ j.\ if\ j = x\ then\ k\ else\ \text{-}\ \$\ j)$ **in** *allE*, *auto*)

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 *hoare-kcomp*:
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 $\equiv (\lambda s.\ if\ P\ s\ then\ X\ s\ else\ Y\ s)$

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

lemma *hoare-if-then-else*:
assumes $(\lambda s.\ P\ s \wedge\ T\ s) \le |X]\ Q$
and $(\lambda s.\ P\ s \wedge \neg\ T\ s) \le |Y]\ Q$
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 $kpower\ f\ n = (\lambda s.\ ((;)\ f\ \hat{\ }\hat{\ }\ n)\ skip\ s)$

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(*force simp*: *subset-antisym*)
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*: $I \le |F]\ I \Longrightarrow I \le |F^*]\ I$
  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 $P \le I$ **and** $I \le Q$ **and** $I \le |F]\ I$
  shows $P \le |F^*]\ Q$
proof−
  have $I \le |F^*]\ I$
    using *assms*(3) *kstar-inv* **by** *blast*
  hence $P \le |F^*]\ I$
    using *assms*(1) **by** *auto*
  also have $|F^*]\ I \le |F^*]\ Q$
    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*: $P \le I \Longrightarrow I \le Q \Longrightarrow I \le |F]\ I \Longrightarrow P \le |LOOP\ F\ INV\ I]\ Q$
  unfolding *loopi-def* **using** *fbox-kstarI*[*of P*] **by** *simp*

lemma *wp-loopI-break*:
  $P \le |Y]\ I \Longrightarrow I \le |X]\ I \Longrightarrow I \le Q \Longrightarrow P \le |Y\ ;\ (LOOP\ X\ INV\ I)]\ Q$
  by (*rule hoare-kcomp*, *force*) (*rule fbox-loopI*, *auto*)

0.4.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

context *local-flow*
begin

lemma *fbox-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
shows $|x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0]\ Q =$
$(\lambda\ s.\ s \in S \longrightarrow (\forall\, t{\in}(U\ s).\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
apply(*unfold fbox-g-orbital fun-eq-iff*)
apply(*clarify*, *rule iffI*; *clarify*)
apply(*force simp*: *in-ivp-sols assms*)
apply(*frule ivp-solsD*(*2*), *frule ivp-solsD*(*3*), *frule ivp-solsD*(*4*))
apply(*subgoal-tac* $\forall \tau{\in}down\ (U\ x)\ t.\ X\ \tau = \varphi\ \tau\ x$)
apply(*clarsimp*, *fastforce*, *rule ballI*)
apply(*rule ivp-unique-solution*[*OF - - - - - in-ivp-sols*])
using *assms* **by** *auto*

lemma *fbox-g-ode*: $|x'=(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0]\ Q =$
$(\lambda s.\ s \in S \longrightarrow (\forall\, t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
by (*subst fbox-g-ode-subset*, *simp-all add*: *init-time interval-time*)

lemma *fbox-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow |x'=(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0]\ Q =$
$(\lambda s.\ s \in S \longrightarrow (\forall\, t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
apply(*subst fbox-g-ode-subset*, *simp-all add*: *subintervalI init-time real-Icc-closed-segment*)
by (*auto simp*: *closed-segment-eq-real-ivl*)

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* :: $(real \Rightarrow ('a{::}banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow$
$real \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow 'a\ set)$ $((1x'{=}\text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}\ DINV\ \text{-}\ ))$
where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$

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

lemma *fbox-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq |x'{=}f\ \&\ G\ on\ U\ S\ @\ t_0]\ I$ **and** $I \leq Q$
shows $P \leq |x'{=}f\ \&\ G\ on\ U\ S\ @\ t_0]\ Q$
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*]:
$(I \leq |x'{=}f\ \&\ G\ on\ U\ S\ @\ t_0]\ I) = \textit{diff-invariant}\ I\ f\ U\ S\ t_0\ G$
by (*auto simp*: *diff-invariant-def ivp-sols-def fbox-def g-orbital-eq*)

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

context *local-flow*
begin

lemma *fbox-diff-inv-eq*:

assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows *diff-invariant* $I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$
$((\lambda s.\ s \in S \longrightarrow I\ s) = |x\acute{}= (\lambda t.\ f)\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ 0]\ (\lambda s.\ s \in S \longrightarrow I\ s))$
unfolding *fbox-diff-inv[symmetric]*
apply(*subst fbox-g-ode-subset[OF assms], simp*)+
apply(*clarsimp simp: le-fun-def fun-eq-iff , safe, force*)
apply(*erule-tac x=0* **in** *ballE*)
using *init-time in-domain ivp(2) assms* **apply**(*force, force*)
apply(*erule-tac x=x* **in** *allE, clarsimp, erule-tac x=t* **in** *ballE*)
using *in-domain ivp(2) assms* **by** *force+*

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

end

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

0.4.3 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.

abbreviation *g-dl-orbit* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a ⇒ 'a set
((1x´=- & -)) **where** $(x\acute{}=f\ \&\ G) \equiv (x\acute{}=(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

abbreviation *g-dl-ode-inv* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a ⇒ 'a set ((1x´=- & - DINV -))
where $(x\acute{}=f\ \&\ G\ DINV\ I) \equiv (x\acute{}=(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0\ DINV\ I)$

lemma *diff-solve-axiom1*:
assumes *local-flow f UNIV UNIV* φ
shows $|x\acute{}= f\ \&\ G]\ Q =$
$(\lambda s.\ \forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
by (*subst local-flow.fbox-g-ode-subset[OF assms], auto*)

lemma *diff-solve-axiom2*:
fixes *c::'a::{heine-borel, banach}*
shows $|x\acute{}=(\lambda s.\ c)\ \&\ G]\ Q =$
$(\lambda s.\ \forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow Q\ (s + t *_R c))$
by (*subst local-flow.fbox-g-ode-subset[OF line-is-local-flow, of UNIV], auto*)

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $P \leq |x\acute{}= f\ \&\ G]\ Q$
using *assms* **by**(*subst local-flow.fbox-g-ode-subset[OF assms(1)]*) *auto*

lemma *diff-weak-axiom1*: $(\ |x\acute{}= f\ \&\ G\ on\ U\ S\ @\ t_0]\ G)\ s$
unfolding *fbox-def g-orbital-eq* **by** *auto*

lemma *diff-weak-axiom2*: $|x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ Q = |x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0]\ (\lambda s.\ G\ s \longrightarrow Q\ s)$
unfolding *fbox-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*: $G \le Q \Longrightarrow P \le |x' = f \ \& \ G \ on \ T \ S \ @ \ t_0] \ Q$
  by(*auto intro*: *g-orbitalD simp*: *le-fun-def g-orbital-eq fbox-def*)

lemma *fbox-g-orbital-eq-univD*:
  assumes $|x' = f \ \& \ G \ on \ U \ S \ @ \ t_0] \ C = (\lambda s. \ True)$
    and $\forall \tau \in (down \ (U \ s) \ t). \ x \ \tau \in (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ s$
  shows $\forall \tau \in (down \ (U \ s) \ t). \ C \ (x \ \tau)$
proof
  fix τ **assume** $\tau \in (down \ (U \ s) \ t)$
  hence $x \ \tau \in (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ s$
    using *assms*(*2*) **by** *blast*
  also have $\forall s'. \ s' \in (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ s \longrightarrow C \ s'$
    using *assms*(*1*) **unfolding** *fbox-def* **by** *meson*
  ultimately show $C \ (x \ \tau)$
    by *blast*
qed

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

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

lemma *diff-inv-axiom1*:
  assumes $G\ s \longrightarrow I\ s$ **and** *diff-invariant* I $(\lambda t.\ f)$ $(\lambda s.\ \{t.\ t \geq 0\})$ *UNIV 0 G*
  shows $(\ |x' = f\ \&\ G]\ I)\ s$
  using *assms* **unfolding** *fbox-g-orbital diff-invariant-eq* **apply** *clarsimp*
  by (*erule-tac x=s* **in** *allE*, *frule ivp-solsD*(*2*), *clarsimp*)

lemma *diff-inv-axiom2*:
  assumes *picard-lindeloef* $(\lambda t.\ f)$ *UNIV UNIV 0*
    and $\bigwedge s.\ \{t{::}real.\ t \geq 0\} \subseteq$ *picard-lindeloef.ex-ivl* $(\lambda t.\ f)$ *UNIV UNIV 0 s*
    and *diff-invariant* I $(\lambda t.\ f)$ $(\lambda s.\ \{t{::}real.\ t \geq 0\})$ *UNIV 0 G*
  shows $|x' = f\ \&\ G]\ I = |(\lambda s.\ \{x.\ s = x \wedge G\ s\})]\ I$
proof(*unfold fbox-g-orbital*, *subst fbox-def*, *clarsimp simp*: *fun-eq-iff*)
  fix *s*
  let *?ex-ivl s* = *picard-lindeloef.ex-ivl* $(\lambda t.\ f)$ *UNIV UNIV 0 s*
  let *?lhs s* =
    $\forall X{\in}Sols\ (\lambda t.\ f)\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ 0\ s.\ \forall t{\geq}0.\ (\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)) \longrightarrow I\ (X\ t)$
  obtain X **where** *xivp1*: $X \in Sols\ (\lambda t.\ f)\ (\lambda s.\ ?ex\text{-}ivl\ s)\ UNIV\ 0\ s$
    using *picard-lindeloef.flow-in-ivp-sols-ex-ivl*[*OF assms*(*1*)] **by** *auto*
  have *xivp2*: $X \in Sols\ (\lambda t.\ f)\ (\lambda s.\ Collect\ ((\leq)\ 0))\ UNIV\ 0\ s$
    by (*rule in-ivp-sols-subset*[*OF - - xivp1*], *simp-all add*: *assms*(*2*))
  hence *shyp*: $X\ 0 = s$
    using *ivp-solsD* **by** *auto*
  have *dinv*: $\forall s.\ I\ s \longrightarrow\ ?lhs\ s$
    using *assms*(*3*) **unfolding** *diff-invariant-eq* **by** *auto*
  {**assume** *?lhs s* **and** *G s*
    hence *I s*
      by (*erule-tac x=X* **in** *ballE*, *erule-tac x=0* **in** *allE*, *auto simp*: *shyp xivp2*)}
  hence $?lhs\ s \longrightarrow (G\ s \longrightarrow I\ s)$
    by *blast*
  moreover
  {**assume** $G\ s \longrightarrow I\ s$
    hence *?lhs s*
      apply(*clarify*, *subgoal-tac* $\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$)
       apply(*erule-tac x=0* **in** *allE*, *frule ivp-solsD*(*2*), *simp*)
      using *dinv* **by** *blast+*}
  ultimately show $?lhs\ s = (G\ s \longrightarrow I\ s)$
    by *blast*
qed

lemma *diff-inv-rule*:
  assumes $P \leq I$ **and** *diff-invariant* $I\ f\ U\ S\ t_0\ G$ **and** $I \leq Q$
  shows $P \leq |x' = f\ \&\ G\ on\ U\ S\ @\ t_0]\ Q$
  apply(*rule fbox-g-orbital-inv*[*OF assms*(*1*) - *assms*(*3*)])
  unfolding *fbox-diff-inv* **using** *assms*(*2*) **.**

end

0.4.4 Examples

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

theory *HS-VC-Examples*
  imports *HS-VC-Spartan*

begin

Pendulum

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

We prove that this motion remains circular.

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

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

— Verified with annotated dynamics.

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

— Verified with differential invariants.

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

— Verified with the flow.

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

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

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

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

— 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 $(x{::}real) \leq h$
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

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

— Verified with annotated dynamics.

lemma *inv-conserv-at-ground*[*bb-real-arith*]:
  assumes *invar*: $2 * g * x = 2 * g * h + v * v$
    and *pos*: $g * \tau^2\ /\ 2 + v * \tau + (x{::}real) = 0$
  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 *inv-conserv-at-air*[*bb-real-arith*]:
  assumes *invar*: $2 * g * x = 2 * g * h + v * v$
  shows $2 * g * (g * \tau^2\ /\ 2 + v * \tau + (x{::}real)) =$
  $2 * g * h + (g * \tau + v) * (g * \tau + v)$ (**is** *?lhs* = *?rhs*)
proof−
  have $?lhs = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
    by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
  also have $... = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
    by(*subst invar*, *simp*)
  finally have *?lhs* = *?middle*.
  moreover
  {**have** $?rhs = g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
    by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
  also have ... = *?middle*
    by (*simp add*: *semiring-normalization-rules*(*29*))
  finally have *?rhs* = *?middle*.}
  ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$(\lambda s.\ s\$1 = h \wedge s\$2 = 0) \leq$
$|LOOP$ (
$(EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T)$;

$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1\ \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$
$(\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h)$
by (*rule fbox-loopI*) (*auto simp*: *bb-real-arith*)

— Verified with the flow.

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

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$(\lambda s.\ s\$1 = h \wedge s\$2 = 0) \leq$
$|LOOP\ ($
$(x' {=} (\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0)\ ;$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1\ \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$
$(\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h)$
apply(*rule fbox-loopI*, *simp-all add*: *local-flow.fbox-g-ode-subset*[*OF local-flow-ball*])
by (*auto 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 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 (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 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* (φ *a L*)
by (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-temp-dyn simp: forall-4 vec-eq-iff*)

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

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

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

lemma *thermostat*:
assumes $a > 0$ **and** $0 \le t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $(\lambda s.\ Tmin \le s\$1 \wedge s\$1 \le Tmax \wedge s\$4 = 0) \le$
$|LOOP$
— control
$((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \le Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \ge Tmax - 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x\acute{}=(\lambda t.\ f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \le -\ (ln\ (Tmin/s\$3))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV$
$@\ 0)$
$ELSE\ (x\acute{}=(\lambda t.\ f\ a\ L)\ \&\ (\lambda s.\ s\$2 \le -\ (ln\ ((L-Tmax)/(L-s\$3)))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0))\ )$
$INV\ (\lambda s.\ Tmin \le s\$1 \wedge s\$1 \le Tmax \wedge (s\$4 = 0 \vee s\$4 = 1))]$
$(\lambda s.\ Tmin \le s\$1 \wedge s\$1 \le Tmax)$
apply(*rule fbox-loopI*, *simp-all add*: *fbox-temp-dyn*[*OF assms*(*1*,*2*)] *le-fun-def*)
using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of* - *Tmax*] **by** *auto*

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

end

0.5 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 *HS-VC-PT*
imports ../*HS-ODEs Transformer-Semantics.Kleisli-Quantale*

begin

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

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

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

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

0.5.1 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*: $P \le Q \Longrightarrow fb_{\mathcal{F}}\ F\ P \le fb_{\mathcal{F}}\ F\ Q$
unfolding *ffb-eq* **by** *auto*

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

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

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

Next, we introduce assignments and their wlps.

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

lemma *vec-upd-eq*: $vec\text{-}upd\ s\ i\ a = (\chi\ j.\ if\ j = i\ then\ a\ else\ s\$j)$
by (*simp add*: *vec-upd-def*)

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*

definition *nondet-assign* :: $'n \Rightarrow 'a\hat{\ }'n \Rightarrow ('a\hat{\ }'n)\ set\ ((2\text{-} ::= ?\ )\ [70]\ 61)$
where $(x ::= ?) = (\lambda s.\ \{(vec\text{-}upd\ s\ x\ k)|k.\ True\})$

lemma *fbox-nondet-assign*[*simp*]: $fb_{\mathcal{F}}\ (x ::= ?)\ P = \{s.\ \forall k.\ (\chi\ j.\ if\ j = x\ then\ k\ else\ s\$j) \in P\}$
unfolding *ffb-eq nondet-assign-def vec-upd-eq* **apply**(*simp add*: *fun-eq-iff*, *safe*)
by (*erule-tac* $x=(\chi\ j.\ if\ j = x\ then\ k\ else\ \text{-}\ \$\ j)$ **in** *allE*, *auto*)

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

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

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

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

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

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

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

We also deal with finite iteration.

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

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

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

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

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

lemma *wp-loopI-break*:
$P \leq fb_{\mathcal{F}}\ Y\ \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ X\ \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (Y\ ;\ (LOOP\ X\ INV\ I))\ Q$
by (*rule hoare-kcomp*, *force*) (*rule ffb-loopI*, *auto*)

0.5.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

lemma *ffb-g-orbital*: $fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ Q =$
$\{s.\ \forall X \in Sols\ f\ U\ S\ t_0\ s.\ \forall t \in U\ s.\ (\forall \tau \in down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow (X\ t) \in Q\}$
unfolding *ffb-eq g-orbital-eq* **by** (*auto simp*: *fun-eq-iff*)

context *local-flow*
begin

lemma *ffb-g-ode-subset*:
  assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
  shows $\textit{fb}_{\mathcal{F}}\ (x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ U\ S\ @\ 0)\ Q =$
  $\{s.\ s \in S \longrightarrow (\forall t \in (U\ s).\ (\forall \tau \in \textit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$
  apply(*unfold ffb-g-orbital set-eq-iff*)
  apply(*clarify*, *rule iffI*; *clarify*)
   apply(*force simp*: *in-ivp-sols assms*)
  apply(*frule ivp-solsD*(*2*), *frule ivp-solsD*(*3*), *frule ivp-solsD*(*4*))
  apply(*subgoal-tac* $\forall \tau \in \textit{down}\ (U\ x)\ t.\ X\ \tau = \varphi\ \tau\ x$)
   apply(*clarsimp*, *fastforce*, *rule ballI*)
  apply(*rule ivp-unique-solution*[*OF - - - - - in-ivp-sols*])
  using *assms* **by** *auto*

lemma *ffb-g-ode*: $\textit{fb}_{\mathcal{F}}\ (x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0)\ Q =$
  $\{s.\ s \in S \longrightarrow (\forall t \in T.\ (\forall \tau \in \textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$ (**is** - = *?wlp*)
  by (*subst ffb-g-ode-subset*, *simp-all add*: *init-time interval-time*)

lemma *ffb-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow \textit{fb}_{\mathcal{F}}\ (x'=(\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ Q =$
  $\{s.\ s \in S \longrightarrow (\forall t \in \{0..t\}.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$
  apply(*subst ffb-g-ode-subset*, *simp-all add*: *subintervalI init-time real-Icc-closed-segment*)
  by (*auto simp*: *closed-segment-eq-real-ivl*)

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

end

Verification with differential invariants

definition *g-ode-inv* :: $(\textit{real} \Rightarrow ('a::\textit{banach}) \Rightarrow 'a) \Rightarrow 'a\ \textit{pred} \Rightarrow ('a \Rightarrow \textit{real set}) \Rightarrow 'a\ \textit{set} \Rightarrow$
  $\textit{real} \Rightarrow 'a\ \textit{pred} \Rightarrow ('a \Rightarrow 'a\ \textit{set})$ ((*1x´=- & - on - - @ - DINV -*))
  where $(x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0\ \textit{DINV}\ I) = (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)$

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

lemma *ffb-g-orbital-inv*:
  assumes $P \leq I$ **and** $I \leq \textit{fb}_{\mathcal{F}}\ (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)\ I$ **and** $I \leq Q$
  shows $P \leq \textit{fb}_{\mathcal{F}}\ (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)\ Q$
  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*]:
  $(\{s.\ I\ s\} \leq \textit{fb}_{\mathcal{F}}\ (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\}) = \textit{diff-invariant}\ I\ f\ U\ S\ t_0\ G$
  by (*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *bdf-diff-inv*:
  $\textit{diff-invariant}\ I\ f\ U\ S\ t_0\ G = (\textit{bd}_{\mathcal{F}}\ (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\} \leq \{s.\ I\ s\})$
  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 $\{s.\ I\ s\} \leq \textit{fb}_{\mathcal{F}}\ (x' = f\ \&\ (\lambda s.\ \textit{True})\ \textit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\}$
  shows $\{s.\ I\ s\} \leq \textit{fb}_{\mathcal{F}}\ (x' = f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\}$
  using *assms* **unfolding** *ffb-diff-inv diff-invariant-eq image-le-pred* **by** *auto*

context *local-flow*

begin

lemma *ffb-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \mathit{is\text{-}interval}\ (U\ s) \wedge U\ s \subseteq T$
shows $\mathit{diff\text{-}invariant}\ I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ \mathit{True}) =$
$(\{s.\ s \in S \longrightarrow I\ s\} = \mathit{fb}_{\mathcal{F}}\ (x\acute{} = (\lambda t.\ f)\ \&\ (\lambda s.\ \mathit{True})\ \mathit{on}\ U\ S\ @\ 0)\ \{s.\ s \in S \longrightarrow I\ s\})$
unfolding *ffb-diff-inv*[*symmetric*]
apply(*subst ffb-g-ode-subset*[*OF assms*], *simp*)+
apply(*clarsimp simp*: *set-eq-iff*, *safe*, *force*)
apply(*erule-tac x=0* **in** *ballE*)
using *init-time in-domain ivp*(*2*) *assms* **apply**(*force*, *force*)
apply(*erule-tac x=x* **in** *allE*, *clarsimp*, *erule-tac x=t* **in** *ballE*)
using *in-domain ivp*(*2*) *assms* **by** *force*+

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

end

lemma *ffb-g-odei*: $P \le \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \le \mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ \mathit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\} \Longrightarrow$
$\{s.\ I\ s \wedge G\ s\} \le Q \Longrightarrow P \le \mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ \mathit{on}\ U\ S\ @\ t_0\ \mathit{DINV}\ I)\ Q$
unfolding *g-ode-inv-def*
apply(*rule-tac b*=$\mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ \mathit{on}\ U\ S\ @\ t_0)\ \{s.\ I\ s\}$ **in** *order.trans*)
apply(*rule-tac I*=$\{s.\ I\ s\}$ **in** *ffb-g-orbital-inv*, *simp-all*)
apply(*subst ffb-g-orbital-guard*, *simp*)
by (*rule ffb-iso*, *force*)

0.5.3 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.

abbreviation *g-dl-orbit* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a ⇒ 'a set ((1x´=- & -))
where $(x\acute{}=f\ \&\ G) \equiv (x\acute{}=(\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ \{t.\ t \ge 0\})\ \mathit{UNIV}\ @\ 0)$

abbreviation *g-dl-ode-inv* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a ⇒ 'a set ((1x´=- & - DINV -))
where $(x\acute{}=f\ \&\ G\ \mathit{DINV}\ I) \equiv (x\acute{}=(\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ \{t.\ t \ge 0\})\ \mathit{UNIV}\ @\ 0\ \mathit{DINV}\ I)$

lemma *diff-solve-axiom1*:
assumes *local-flow f UNIV UNIV* φ
shows $\mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G)\ Q =$
$\{s.\ \forall t \ge 0.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q\}$
by (*subst local-flow.ffb-g-ode-subset*[*OF assms*], *auto*)

lemma *diff-solve-axiom2*:
fixes *c*::'a::{*heine-borel*, *banach*}
shows $\mathit{fb}_{\mathcal{F}}\ (x\acute{}=(\lambda s.\ c)\ \&\ G)\ Q =$
$\{s.\ \forall t \ge 0.\ (\forall \tau \in \{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow (s + t *_R c) \in Q\}$
apply(*subst local-flow.ffb-g-ode-subset*[**where** $\varphi=(\lambda t\ s.\ s + t *_R c)$ **and** *T=UNIV*])
by (*rule line-is-local-flow*, *auto*)

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ s \in P \longrightarrow (\forall t \ge 0.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)$
shows $P \le \mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G)\ Q$
using *assms* **by**(*subst local-flow.ffb-g-ode-subset*[*OF assms*(*1*)]) *auto*

lemma *diff-weak-axiom1*: $s \in (\mathit{fb}_{\mathcal{F}}\ (x\acute{} = f\ \&\ G\ \mathit{on}\ U\ S\ @\ t_0)\ \{s.\ G\ s\})$
unfolding *ffb-eq g-orbital-eq* **by** *auto*

lemma *diff-weak-axiom2*: $fb_{\mathcal{F}}$ $(x' = f$ & 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* U S @ $t_0)$ $\{s.\ C\ s\}$ = *UNIV*
    and $\forall \tau \in (down\ (U\ s)\ t).\ x\ \tau \in (x' = f$ & G *on* U S @ $t_0)$ s
  shows $\forall \tau \in (down\ (U\ s)\ t).\ C\ (x\ \tau)$
proof
  fix τ **assume** $\tau \in (down\ (U\ s)\ t)$
  hence $x\ \tau \in (x' = f$ & G *on* U S @ $t_0)$ s
    using *assms*(*2*) **by** *blast*
  also have $\forall y.\ y \in (x' = f$ & G *on* U S @ $t_0)$ $s \longrightarrow C\ y$
    using *assms*(*1*) **unfolding** *ffb-eq* **by** *fastforce*
  ultimately show $C\ (x\ \tau)$ **by** *blast*
qed

lemma *diff-cut-axiom*:
  assumes $fb_{\mathcal{F}}$ $(x' = f$ & G *on* U S @ $t_0)$ $\{s.\ C\ s\}$ = *UNIV*
  shows $fb_{\mathcal{F}}$ $(x' = f$ & G *on* U S @ $t_0)$ Q = $fb_{\mathcal{F}}$ $(x' = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* U 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* U S @ $t_0)$ s
    then obtain τ::*real* **and** X **where** *x-ivp*: $X \in$ *Sols* f U S t_0 s
      and $X\ \tau = s'$ **and** $\tau \in (U\ s)$ **and** *guard-x*:$\mathcal{P}$ X $(down\ (U\ s)\ \tau) \subseteq \{s.\ G\ s\}$
      using *g-orbitalD*[*of* s' f G - S t_0 s] **by** *blast*
    have $\forall t \in (down\ (U\ s)\ \tau).\ \mathcal{P}\ X\ (down\ (U\ s)\ t) \subseteq \{s.\ G\ s\}$
      using *guard-x* **by** (*force simp*: *image-def*)
    also have $\forall t \in (down\ (U\ s)\ \tau).\ t \in (U\ s)$
      using ‹$\tau \in (U\ s)$› *closed-segment-subset-interval* **by** *auto*
    ultimately have $\forall t \in (down\ (U\ s)\ \tau).\ X\ t \in (x' = f$ & G *on* U S @ $t_0)$ s
      using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
    hence $\forall t \in (down\ (U\ s)\ \tau).\ C\ (X\ t)$
      using *assms* **unfolding** *ffb-eq* **by** *fastforce*
    hence $s' \in (x' = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* U S @ $t_0)$ s
      using *g-orbitalI*[*OF x-ivp* ‹$\tau \in (U\ s)$›] *guard-x* ‹$X\ \tau = s'$›
      unfolding *image-le-pred* **by** *fastforce***}**
  thus $(x' = f$ & G *on* U S @ $t_0)$ $s \subseteq (x' = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* U S @ $t_0)$ s
    by *blast*
next show $\bigwedge s.\ (x' = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* U S @ $t_0)$ $s \subseteq (x' = f$ & G *on* U S @ $t_0)$ s
    by (*auto simp*: *g-orbital-eq*)
qed

lemma *diff-cut-rule*:
  assumes *ffb-C*: $P \leq fb_{\mathcal{F}}$ $(x' = f$ & G *on* U S @ $t_0)$ $\{s.\ C\ s\}$
    and *ffb-Q*: $P \leq fb_{\mathcal{F}}$ $(x' = f$ & $(\lambda s.\ G\ s \wedge C\ s)$ *on* U S @ $t_0)$ Q
  shows $P \leq fb_{\mathcal{F}}$ $(x' = f$ & G *on* U 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** $t \in (U\ s)$
    and *x-ivp*:$X \in$ *Sols* f U S t_0 s
    and *guard-x*:$\forall \tau.\ \tau \in (U\ s) \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
  have $\forall \tau \in (down\ (U\ s)\ t).\ X\ \tau \in (x' = f$ & G *on* U S @ $t_0)$ s
    using *g-orbitalI*[*OF x-ivp*] *guard-x* **unfolding** *image-le-pred* **by** *auto*
  hence $\forall \tau \in (down\ (U\ s)\ t).\ C\ (X\ \tau)$
    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* U S @ $t_0)$ s

```
    using guard-x ‹t ∈ (U s)› by (auto intro!: g-orbitalI x-ivp)
  thus (X t) ∈ Q
    using ‹s ∈ P› ffb-Q by (subst (asm) ffb-eq) auto
qed

lemma diff-inv-axiom1:
  assumes G s ⟶ I s and diff-invariant I (λt. f) (λs. {t. t ≥ 0}) UNIV 0 G
  shows s ∈ (fb𝓕 (x´= f & G) {s. I s})
  using assms unfolding ffb-g-orbital diff-invariant-eq apply clarsimp
  by (erule-tac x=s in allE, frule ivp-solsD(2), clarsimp)

lemma diff-inv-axiom2:
  assumes picard-lindeloef (λt. f) UNIV UNIV 0
    and ⋀s. {t::real. t ≥ 0} ⊆ picard-lindeloef.ex-ivl (λt. f) UNIV UNIV 0 s
    and diff-invariant I (λt. f) (λs. {t::real. t ≥ 0}) UNIV 0 G
  shows fb𝓕 (x´= f & G) {s. I s} = fb𝓕 (λs. {x. s = x ∧ G s}) {s. I s}
proof(unfold ffb-g-orbital, subst ffb-eq, clarsimp simp: set-eq-iff)
  fix s
  let ?ex-ivl s = picard-lindeloef.ex-ivl (λt. f) UNIV UNIV 0 s
  let ?lhs s =
    ∀X∈Sols (λt. f) (λs. {t. t ≥ 0}) UNIV 0 s. ∀t≥0. (∀τ. 0 ≤ τ ∧ τ ≤ t ⟶ G (X τ)) ⟶ I (X t)
  obtain X where xivp1: X ∈ Sols (λt. f) (λs. ?ex-ivl s) UNIV 0 s
    using picard-lindeloef.flow-in-ivp-sols-ex-ivl[OF assms(1)] by auto
  have xivp2: X ∈ Sols (λt. f) (λs. Collect ((≤) 0)) UNIV 0 s
    by (rule in-ivp-sols-subset[OF - - xivp1], simp-all add: assms(2))
  hence shyp: X 0 = s
    using ivp-solsD by auto
  have dinv: ∀s. I s ⟶ ?lhs s
    using assms(3) unfolding diff-invariant-eq by auto
  {assume ?lhs s and G s
    hence I s
      by (erule-tac x=X in ballE, erule-tac x=0 in allE, auto simp: shyp xivp2)}
  hence ?lhs s ⟶ (G s ⟶ I s)
    by blast
  moreover
  {assume G s ⟶ I s
    hence ?lhs s
      apply(clarify, subgoal-tac ∀τ. 0 ≤ τ ∧ τ ≤ t ⟶ G (X τ))
       apply(erule-tac x=0 in allE, frule ivp-solsD(2), simp)
      using dinv by blast+}
  ultimately show ?lhs s = (G s ⟶ I s)
    by blast
qed

lemma diff-inv-rule:
  assumes P ≤ {s. I s} and diff-invariant I f U S t₀ G and {s. I s} ≤ Q
  shows P ≤ fb𝓕 (x´= f & G on U S @ t₀) Q
  apply(rule ffb-g-orbital-inv[OF assms(1) - assms(3)])
  unfolding ffb-diff-inv using assms(2) .

end
```

0.5.4 Examples

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

```
theory HS-VC-PT-Examples
  imports HS-VC-PT
```

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

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

— Verified by providing the dynamics

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

— Verified with differential invariants.

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

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 $f\ g\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ g)$

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

— 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 $(x{::}real) \leq h$
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

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
  $\{s.\ s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$
  $(LOOP\ ($
    $(x' = (f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0))$;
    $(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ - s\$2))\ ELSE\ skip))$
  $INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0))$
  $\{s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
  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 by providing the dynamics

lemma *inv-conserv-at-ground*[*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 *inv-conserv-at-air*[*bb-real-arith*]:
  assumes *invar*: $2 * g * x = 2 * g * h + v * v$
  shows $2 * g * (g * \tau^2 / 2 + v * \tau + (x::real)) =$
  $2 * g * h + (g * \tau + v) * (g * \tau + v)$ (**is** *?lhs* = *?rhs*)
proof−
  have $?lhs = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
    by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
  also have $... = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
    by(*subst invar*, *simp*)
  finally have *?lhs* = *?middle*.
  moreover
  {**have** $?rhs = g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
    by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
  also have ... = *?middle*
    by (*simp add*: *semiring-normalization-rules*(*29*))

finally have *?rhs = ?middle*.}
ultimately show *?thesis* **by** *auto*
qed

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

— Verified with the flow.

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

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$
$(LOOP\ ($
$(x' = (\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0)\ ;$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\{s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
by (*rule ffb-loopI*) (*auto simp*: *bb-real-arith local-flow.ffb-g-ode*[*OF local-flow-ball*])

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 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*ˆ*4* $\Rightarrow$ *real*ˆ*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*ˆ*4* $\Rightarrow$ *real*ˆ*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*: $0 < Tmin\ Tmin \le T\ T \le Tmax$
and *thyps*: $0 \le (t::real)\ \forall\tau\in\{0..t\}.\ \tau \le - (ln\ (Tmin\ /\ T)\ /\ a)$
shows $Tmin \le exp\ (-a * t) * T$ **and** $exp\ (-a * t) * T \le Tmax$
proof−
have $0 \le t \wedge t \le - (ln\ (Tmin\ /\ T)\ /\ a)$
using *thyps* **by** *auto*
hence $ln\ (Tmin\ /\ T) \le - a * t \wedge - a * t \le 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have $Tmin\ /\ T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $Tmin\ /\ T \le exp\ (-a * t)\ exp\ (-a * t) \le 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
thus $Tmin \le exp\ (-a * t) * T$
using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
show $exp\ (-a * t) * T \le Tmax$
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 *temp-dyn-up-real-arith*:
assumes $a > 0$ **and** *Thyps*: $Tmin \le T\ T \le Tmax\ Tmax < (L::real)$
and *thyps*: $0 \le t\ \forall\tau\in\{0..t\}.\ \tau \le - (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
shows $L - Tmax \le exp\ (-(a * t)) * (L - T)$
and $L - exp\ (-(a * t)) * (L - T) \le Tmax$
and $Tmin \le L - exp\ (-(a * t)) * (L - T)$
proof−
have $0 \le t \wedge t \le - (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
using *thyps* **by** *auto*
hence $ln\ ((L - Tmax)\ /\ (L - T)) \le - a * t \wedge - a * t \le 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have $(L - Tmax)\ /\ (L - T) > 0$
using *Thyps* **by** *auto*
ultimately have $(L - Tmax)\ /\ (L - T) \le exp\ (-a * t) \wedge exp\ (-a * t) \le 1$
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*: $(L - Tmax) \le exp\ (-a * t) * (L - T) \wedge exp\ (-a * t) * (L - T) \le (L - T)$
by (*simp add*: *pos-divide-le-eq*)
thus $(L - Tmax) \le exp\ (-(a * t)) * (L - T)$
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 *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 $\{s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0\} \leq fb_{\mathcal{F}}$
(*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
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x\acute{} =(\lambda t.\ f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV$
$@\ 0)$
$ELSE\ (x\acute{} =(\lambda t.\ f\ a\ L)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ ((L-Tmax)/(L-s\$3)))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0))\ )$
$INV\ (\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
$\{s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax\}$
apply(*rule ffb-loopI*, *simp-all add*: *ffb-temp-dyn*[*OF assms*(*1*,*2*)] *le-fun-def*, *safe*)
using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of - Tmax*] **by** *auto*

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

end

0.6 Verification components with MKA

We use the forward box operator of antidomain Kleene algebras to derive rules for weakest liberal preconditions (wlps) of regular programs.

theory *HS-VC-MKA*
imports *KAD.Modal-Kleene-Algebra*

begin

0.6.1 Verification in AKA

Here we derive verification components with weakest liberal preconditions based on antidomain Kleene algebra

no-notation *Range-Semiring.antirange-semiring-class.ars-r* (*r*)
and *HOL.If* ((*if* (-)/ *then* (-)/ *else* (-)) [*0*, *0*, *10*] *10*)

notation *zero-class.zero* (*0*)

context *antidomain-kleene-algebra*
begin

— Skip

lemma $|1]\ x = d\ x$
using *fbox-one* **.**

— Abort

lemma $|0]\ q = 1$
  using *fbox-zero* **.**

— Sequential composition

lemma $|x \cdot y]\ q = |x]\ |y]\ q$
  using *fbox-mult* **.**

declare *fbox-mult* [*simp*]

— Nondeterministic choice

lemma $|x + y]\ q = |x]\ q \cdot |y]\ q$
  using *fbox-add2* **.**

lemma *le-fbox-choice-iff*: $d\ p \leq |x + y]q \longleftrightarrow (d\ p \leq |x]q) \wedge (d\ p \leq |y]q)$
  by (*metis local.a-closure′ local.ads-d-def local.dnsz.dom-glb-eq local.fbox-add2 local.fbox-def*)

— Conditional statement

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

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*$]\ q = (ad\ p + |x]\ q) \cdot (d\ p + |y]\ q)$
  using *fbox-export1 local.ans-d-def local.fbox-mult*
  unfolding *aka-cond-def ads-d-def fbox-def* **by** *auto*

lemma *fbox-cond2*: $|$*if p then x else y*$]\ q = (d\ p \cdot |x]\ q) + (ad\ p \cdot |y]\ q)$ (**is** $?lhs = ?d1 + ?d2$)
proof −
  have *obs*: $?lhs = d\ p \cdot ?lhs + ad\ p \cdot ?lhs$
    by (*metis* (*no-types*, *lifting*) *local.a-closure′ local.a-de-morgan fbox-def ans-d-def*
      ads-d-def local.am2 local.am5-lem local.dka.dsg3 local.dka.dsr5)
  have $d\ p \cdot ?lhs = d\ p \cdot |x]\ q \cdot (d\ p + d\ (\ |y]\ q))$
    using *fbox-cond local.a-d-add-closure local.ads-d-def*
    local.ds.ddual.mult-assoc local.fbox-def **by** *auto*
  also have $... = d\ p \cdot |x]\ q$
    by (*metis local.ads-d-def local.am2 local.dka.dns5 local.ds.ddual.mult-assoc local.fbox-def*)
  finally have $d\ p \cdot ?lhs = d\ p \cdot |x]\ q$ **.**
  moreover have $ad\ p \cdot ?lhs = ad\ p \cdot |y]\ q$
    by (*metis add-commute fbox-cond local.a-closure′ local.a-mult-add ads-d-def ans-d-def*
      local.dnsz.dns5 local.ds.ddual.mult-assoc local.fbox-def)
  ultimately show *?thesis*
    using *obs* **by** *simp*
qed

— While loop

definition *aka-whilei* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - inv -* [64,64,64] 63) **where**
  while t do x inv i $= (d\ t \cdot x)^\star \cdot ad\ t$

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 *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*)

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-while*: $d\ p \cdot d\ t \leq |x]\ p \Longrightarrow d\ p \leq |(d\ t \cdot x)^\star \cdot ad\ t]\ (d\ p \cdot ad\ t)$
proof –
assume $d\ p \cdot d\ t \leq |x]\ p$
hence $d\ p \leq |d\ t \cdot x]\ p$
by (*simp add*: *fbox-export1 fbox-shunt*)
hence $d\ p \leq |(d\ t \cdot x)^\star]\ p$
by (*simp add*: *fbox-star-induct-var*)
thus *?thesis*
using *order-trans fbox-export2* **by** *presburger*
qed

lemma *fbox-whilei*:
assumes $d\ p \leq d\ i$ **and** $d\ i \cdot ad\ t \leq d\ q$ **and** $d\ i \cdot d\ t \leq |x]\ i$
shows $d\ p \leq |while\ t\ do\ x\ inv\ i]\ q$
proof–
have $d\ i \leq |(d\ t \cdot x)^\star \cdot ad\ t]\ (d\ i \cdot ad\ t)$
using *fbox-while assms* **by** *blast*
also have ... $\leq |(d\ t \cdot x)^\star \cdot ad\ t]\ q$
by (*metis assms*(*2*) *local.dka.dom-iso local.dka.domain-invol local.fbox-iso*)
finally show *?thesis*
unfolding *aka-whilei-def*
using *assms*(*1*) *local.dual-order.trans* **by** *blast*
qed

lemma *fbox-seq-var*: $p \leq |x]\ p' \Longrightarrow p' \leq |y]\ q \Longrightarrow\ p \leq |x \cdot y]\ q$
proof –
assume *h1*: $p \leq |x]\ p'$ **and** *h2*: $p' \leq |y]\ q$
hence $|x]\ p' \leq |x]\ |y]\ q$
by (*metis ads-d-def fbox-antitone-var fbox-dom fbox-iso*)
thus *?thesis*
by (*metis dual-order.trans fbox-mult h1*)
qed

lemma *fbox-whilei-break*:
$d\ p \leq |y]\ i \Longrightarrow d\ i \cdot ad\ t \leq d\ q \Longrightarrow d\ i \cdot d\ t \leq |x]\ i \Longrightarrow d\ p \leq |y \cdot (while\ t\ do\ x\ inv\ i)]\ q$
apply (*rule fbox-seq-var*[*OF - fbox-whilei*])
using *fbox-simp* **by** *auto*

— Finite iteration

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

lemma $d\ p \leq |x]\ p \Longrightarrow d\ p \leq |x^\star]\ p$
using *fbox-star-induct-var* **.**

lemma *fbox-loopi*: $p \leq d\ i \Longrightarrow d\ i \leq |x]\ i \Longrightarrow d\ i \leq d\ q \Longrightarrow p \leq |loop\ x\ inv\ i]\ q$
unfolding *aka-loopi-def* **by** (*meson dual-order.trans fbox-iso fbox-star-induct-var*)

lemma *fbox-loopi-break*:
$p \le |y]\ d\ i \Longrightarrow d\ i \le |x]\ i \Longrightarrow d\ i \le d\ q \Longrightarrow p \le |y \cdot (loop\ x\ inv\ i)]\ q$
by (*rule fbox-seq-var*, *force*) (*rule fbox-loopi*, *auto*)

— Invariants

lemma $p \le i \Longrightarrow i \le |x]i \Longrightarrow i \le q \Longrightarrow p \le |x]q$
by (*metis local.ads-d-def local.dpdz.dom-iso local.dual-order.trans local.fbox-iso*)

lemma $p \le d\ i \Longrightarrow d\ i \le |x]i \Longrightarrow i \le d\ q \Longrightarrow p \le |x]q$
by (*metis local.a-4 local.a-antitone′ local.a-subid-aux2 ads-d-def local.antisym fbox-def local.dka.dsg1 local.dual.mult-isol-var local.dual-order.trans local.order.refl*)

lemma $(i \le |x]\ i) \vee (j \le |x]\ j) \Longrightarrow (i + j) \le |x]\ (i + j)$

oops

lemma $d\ i \le |x]\ i \Longrightarrow d\ j \le |x]\ j \Longrightarrow (d\ i + d\ j) \le |x]\ (d\ i + d\ j)$
by (*metis* (*no-types*, *lifting*) *dual-order.trans fbox-simp fbox-subdist join.le-supE join.le-supI*)

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

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

end

end

theory *HS-VC-MKA-rel*
imports *../HS-ODEs HS-VC-MKA*

begin

We follow Gomes and Struth in showing that relations form an antidomain Kleene algebra. These allows us to inherit the rules of the wlp calculus for regular programs. Finally, we derive three methods for verifying correctness specifications for the continuous dynamics of hybris systems in this setting.

0.6.2 Relational model

context *dioid-one-zero*
begin

lemma *power-inductl*: $z + x \cdot y \le y \Longrightarrow (x \hat{\ } n) \cdot z \le y$
by(*induct n*, *auto*, *metis mult.assoc mult-isol order-trans*)

lemma *power-inductr*: $z + y \cdot x \le y \Longrightarrow z \cdot (x \hat{\ } n) \le y$
proof (*induct n*)
case *0* **show** *?case*
using *0.prems* **by** *auto*
case *Suc*
{**fix** *n*
assume $z + y \cdot x \le y \Longrightarrow z \cdot x \hat{\ } n \le y$
and $z + y \cdot x \le y$
hence $z \cdot x \hat{\ } n \le y$

```
      by auto
    also have z · x ^ Suc n = z · x · x ^ n
      by (metis mult.assoc power-Suc)
    moreover have ... = (z · x ^ n) · x
      by (metis mult.assoc power-commutes)
    moreover have ... ≤ y · x
      by (metis calculation(1) mult-isor)
    moreover have ... ≤ y
      using ‹z + y · x ≤ y› by auto
    ultimately have z · x ^ Suc n ≤ y by auto}
  thus ?case
    by (metis Suc)
qed

end

interpretation rel-dioid: dioid-one-zero (∪) (O) Id {} (⊆) (⊂)
  by (unfold-locales, auto)

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^* = (⋃ n. rel-dioid.power X n)
  by (simp add: power-is-relpow rtrancl-is-UN-relpow)

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

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

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

definition rel-ad :: 'a rel ⇒ 'a rel where
  rel-ad R = {(x,x) | x. ¬ (∃ y. (x,y) ∈ R)}

interpretation rel-aka: antidomain-kleene-algebra rel-ad (∪) (O) Id {} (⊆) (⊂) rtrancl
  by unfold-locales (auto simp: rel-ad-def)
```

0.6.3 Store and weakest preconditions

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

no-notation *Archimedean-Field.ceiling* $(\lceil\text{-}\rceil)$
and *antidomain-semiringl.ads-d* (d)

notation *Id* $(skip)$
and *relcomp* $($**infixl** $;\ 70)$
and *zero-class.zero* (0)
and *rel-aka.fbox* (wp)

definition $p2r :: 'a\ pred \Rightarrow 'a\ rel\ ((1\lceil\text{-}\rceil))$ **where**
$\lceil P\rceil = \{(s,s)\ |s.\ P\ s\}$

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

lemma *in-p2r* [*simp*]: $(a,b) \in \lceil P\rceil = (P\ a \wedge a = b)$
by (*auto simp*: *p2r-def*)

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

lemma *wp-test*[*simp*]: $wp\ \lceil P\rceil\ \lceil Q\rceil = \lceil \lambda s.\ P\ s \longrightarrow Q\ s\rceil$
by (*subst wp-rel*, *simp add*: *p2r-def*)

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

lemma *vec-upd-eq*: $vec\text{-}upd\ s\ i\ a = (\chi\ j.\ if\ j = i\ then\ a\ else\ s\$j)$
by (*simp add*: *vec-upd-def*)

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

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*)

definition $nondet\text{-}assign :: 'b \Rightarrow ('a\hat{}'b)\ rel\ ((2\text{-}\ ::=\ ?\ )\ [70]\ 61)$
where $(x ::= ?) = \{(s,vec\text{-}upd\ s\ x\ k)|s\ k.\ True\}$

lemma *wp-nondet-assign*[*simp*]: $wp\ (x ::= ?)\ \lceil P\rceil = \lceil \lambda s.\ \forall k.\ P\ (\chi\ j.\ (((\$)\ s)(x := k))\ j)\rceil$
unfolding *wp-rel nondet-assign-def vec-upd-eq* **apply**(*clarsimp*, *safe*)
by (*erule-tac* $x=(\chi\ j.\ if\ j = x\ then\ k\ else\ s\ \$\ j)$ **in** *allE*, *auto*)

lemma *le-wp-choice-iff*: $\lceil P\rceil \leq wp\ (X \cup Y)\ \lceil Q\rceil \longleftrightarrow \lceil P\rceil \leq wp\ X\ \lceil Q\rceil \wedge \lceil P\rceil \leq wp\ Y\ \lceil Q\rceil$
using *rel-aka.le-fbox-choice-iff*[*of* $\lceil P\rceil$] **by** *simp*

abbreviation $cond\text{-}sugar :: 'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel\ (IF\ \text{-}\ THEN\ \text{-}\ ELSE\ \text{-}\ [64,64]\ 63)$
where $IF\ P\ THEN\ X\ ELSE\ Y \equiv rel\text{-}aka.aka\text{-}cond\ \lceil P\rceil\ X\ Y$

abbreviation $loopi\text{-}sugar :: 'a\ rel \Rightarrow 'a\ pred \Rightarrow 'a\ rel\ (LOOP\ \text{-}\ INV\ \text{-}\ \ [64,64]\ 63)$
where $LOOP\ R\ INV\ I \equiv rel\text{-}aka.aka\text{-}loopi\ R\ \lceil I\rceil$

lemma *change-loopI*: $LOOP\ X\ INV\ G = LOOP\ X\ INV\ I$
by (*unfold rel-aka.aka-loopi-def*, *simp*)

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*

lemma *wp-loopI-break*:
$\lceil P\rceil \leq wp\ Y\ \lceil I\rceil \Longrightarrow \lceil I\rceil \leq wp\ X\ \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow \lceil P\rceil \leq wp\ (Y\ ;\ (LOOP\ X\ INV\ I))\ \lceil Q\rceil$
using *rel-aka.fbox-loopi-break*[*of* $\lceil P\rceil$] **by** *auto*

0.6.4 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

definition *g-ode* :: $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow real \Rightarrow$
$'a\ rel$ ((1x´=- & - on - - @ -))
where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0) = \{(s,s')\ |s\ s'.\ s' \in g\text{-}orbital\ f\ G\ U\ S\ t_0\ s\}$

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

context *local-flow*
begin

lemma *wp-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \land is\text{-}interval\ (U\ s) \land U\ s \subseteq T$
shows $wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \lceil Q\rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}(U\ s).\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
apply(*unfold wp-g-orbital*, *clarsimp*, *rule iffI*; *clarify*)
apply(*force simp*: *in-ivp-sols assms*)
apply(*frule ivp-solsD*(*2*), *frule ivp-solsD*(*3*), *frule ivp-solsD*(*4*))
apply(*subgoal-tac* $\forall \tau{\in}down\ (U\ s)\ t.\ X\ \tau = \varphi\ \tau\ s$)
apply(*clarsimp*, *fastforce*, *rule ballI*)
apply(*rule ivp-unique-solution*[*OF - - - - - in-ivp-sols*])
using *assms* **by** *auto*

lemma *wp-g-ode*: $wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ \lceil Q\rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
by (*subst wp-g-ode-subset*, *simp-all add*: *init-time interval-time*)

lemma *wp-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ \lceil Q\rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
apply(*subst wp-g-ode-subset*, *simp-all add*: *subintervalI init-time real-Icc-closed-segment*)
by (*auto simp*: *closed-segment-eq-real-ivl*)

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

end

Verification with differential invariants

definition *g-ode-inv* :: $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow$
$real \Rightarrow 'a\ pred \Rightarrow 'a\ rel$ $((1x' = \text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}\ DINV\ \text{-}))$
where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$

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

lemma *wp-g-orbital-inv*:
assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
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*]: $(\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil) = diff\text{-}invariant\ I\ f\ U\ S\ t_0\ G$
unfolding *diff-invariant-eq wp-g-orbital* **by**(*auto simp*: *p2r-def*)

lemma *diff-inv-guard-ignore*:
assumes $\lceil I \rceil \leq wp\ (x' = f\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ t_0)\ \lceil I \rceil$
shows $\lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil$
using *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*
begin

lemma *wp-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
shows $diff\text{-}invariant\ I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$
$(\lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil = wp\ (x' = (\lambda t.\ f)\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ 0)\ \lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil)$
unfolding *wp-diff-inv*[*symmetric*]
apply(*subst wp-g-ode-subset*[*OF assms*], *simp*)+
apply(*clarsimp*, *safe*, *force*)
apply(*erule-tac x=0* **in** *ballE*)
using *init-time in-domain ivp*(*2*) *assms* **apply**(*force*, *force*)
apply(*erule-tac x=s* **in** *allE*, *clarsimp*, *erule-tac x=t* **in** *ballE*)
using *in-domain ivp*(*2*) *assms* **by** *force+*

lemma *diff-inv-eq-inv-set*:
$diff\text{-}invariant\ I\ (\lambda t.\ f)\ (\lambda s.\ T)\ S\ 0\ (\lambda s.\ True) = (\forall s.\ I\ s \longrightarrow \gamma^{\varphi}\ s \subseteq \{s.\ I\ s\})$
unfolding *diff-inv-eq-inv-set orbit-def* **by** (*auto simp*: *p2r-def*)

end

lemma *wp-g-odei*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I)\ \lceil Q \rceil$
unfolding *g-ode-inv-def*
apply(*rule-tac b*=$wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil$ **in** *order.trans*)
apply(*rule-tac I=I* **in** *wp-g-orbital-inv*, *simp-all*)
apply(*subst wp-g-orbital-guard*, *simp*)
by (*rule rel-aka.fbox-iso*, *simp*)

0.6.5 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.

abbreviation *g-dl-ode* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a rel ((1x´=- & -))
  where $(x'{=}f\ \&\ G) \equiv (x'{=}\ (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

abbreviation *g-dl-ode-inv* :: (('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a rel ((1x´=- & - DINV -))
  where $(x'{=}\ f\ \&\ G\ DINV\ I) \equiv (x'{=}\ (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0\ DINV\ I)$

lemma *diff-solve-axiom1*:
  assumes *local-flow f UNIV UNIV* φ
  shows $wp\ (x'{=}\ f\ \&\ G)\ \lceil Q\rceil =$
  $\lceil\lambda s.\ \forall t{\geq}0.\ (\forall\tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$
  by (*subst local-flow.wp-g-ode-subset*[*OF assms*], *auto*)

lemma *diff-solve-axiom2*:
  fixes *c*::'a::{*heine-borel*, *banach*}
  shows $wp\ (x'{=}(\lambda s.\ c)\ \&\ G)\ \lceil Q\rceil =$
  $\lceil\lambda s.\ \forall t{\geq}0.\ (\forall\tau{\in}\{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow Q\ (s + t *_R c)\rceil$
  apply(*subst local-flow.wp-g-ode-subset*[**where** $\varphi{=}(\lambda t\ s.\ s + t *_R c)$ **and** *T=UNIV*])
  by (*rule line-is-local-flow*, *auto*)

lemma *diff-solve-rule*:
  assumes *local-flow f UNIV UNIV* φ
    and $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall\tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  shows $\lceil P\rceil \leq wp\ (x'{=}\ f\ \&\ G)\ \lceil Q\rceil$
  using *assms* **by** (*subst local-flow.wp-g-ode-subset*[*OF assms*(*1*)], *auto*)

lemma *diff-weak-axiom1*: $Id \subseteq wp\ (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil G\rceil$
  unfolding *wp-rel g-ode-def g-orbital-eq* **by** *auto*

lemma *diff-weak-axiom2*:
  $wp\ (x'{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil = wp\ (x'{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil\lambda\ s.\ G\ s \longrightarrow Q\ s\rceil$
  unfolding *wp-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*:
  assumes $\lceil G\rceil \leq \lceil Q\rceil$
  shows $\lceil P\rceil \leq wp\ (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$
  using *assms* **by** (*auto simp*: *wp-g-orbital*)

lemma *wp-g-evol-IdD*:
  assumes $wp\ (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C\rceil = Id$
    and $\forall\tau{\in}(down\ (U\ s)\ t).\ (s,\ x\ \tau) \in (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)$
  shows $\forall\tau{\in}(down\ (U\ s)\ t).\ C\ (x\ \tau)$
proof
  fix τ **assume** $\tau \in (down\ (U\ s)\ t)$
  hence $x\ \tau \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
    using *assms*(*2*) **unfolding** *g-ode-def* **by** *blast*
  also have $\forall y.\ y \in ($*g-orbital* $f\ G\ U\ S\ t_0\ s) \longrightarrow C\ y$
    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 $wp\ (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C\rceil = Id$
  shows $wp\ (x'{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil = wp\ (x'{=}\ f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$
proof(*rule-tac f=*$\lambda\ x.\ wp\ x\ \lceil Q\rceil$ **in** *HOL.arg-cong*, *rule subset-antisym*)

```
  show (x´=f & G on U S @ t_0) ⊆ (x´=f & λs. G s ∧ C s on U S @ t_0)
  proof(clarsimp simp: g-ode-def)
    fix s and s′ assume s′ ∈ g-orbital f G U S t_0 s
    then obtain τ::real and X where x-ivp: X ∈ Sols f U S t_0 s
      and X τ = s′ and τ ∈ U s and guard-x:(𝒫 X (down (U s) τ) ⊆ {s. G s})
      using g-orbitalD[of s′ f G - S t_0 s] by blast
    have ∀ t∈(down (U s) τ). 𝒫 X (down (U s) t) ⊆ {s. G s}
      using guard-x by (force simp: image-def)
    also have ∀ t∈(down (U s) τ). t ∈ U s
      using ‹τ ∈ U s› by auto
    ultimately have ∀ t∈(down (U s) τ). X t ∈ g-orbital f G U S t_0 s
      using g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))
    hence ∀ t∈(down (U s) τ). C (X t)
      using wp-g-evol-IdD[OF assms(1)] unfolding g-ode-def by blast
    thus s′ ∈ g-orbital f (λs. G s ∧ C s) U S t_0 s
      using g-orbitalI[OF x-ivp ‹τ ∈ U s›] guard-x ‹X τ = s′› by fastforce
  qed
next show (x´=f & λs. G s ∧ C s on U S @ t_0) ⊆ (x´=f & G on U S @ t_0)
    by (auto simp: g-orbital-eq g-ode-def)
qed

lemma diff-cut-rule:
  assumes wp-C: ⌈P⌉ ≤ wp (x´= f & G on U S @ t_0) ⌈C⌉
    and wp-Q: ⌈P⌉ ≤ wp (x´= f & (λs. G s ∧ C s) on U S @ t_0) ⌈Q⌉
  shows ⌈P⌉ ≤ wp (x´= f & G on U S @ t_0) ⌈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 ∈ U s
    and x-ivp:X ∈ Sols f U S t_0 s
    and guard-x:∀ x. x ∈ U s ∧ x ≤ t ⟶ G (X x)
  have ∀ t∈(down (U s) t). X t ∈ g-orbital f G U S t_0 s
    using g-orbitalI[OF x-ivp] guard-x by auto
  hence ∀ t∈(down (U s) 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) U S t_0 s
    using guard-x ‹t ∈ U s› 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

lemma diff-inv-axiom1:
  assumes G s ⟶ I s and diff-invariant I (λt. f) (λs. {t. t ≥ 0}) UNIV 0 G
  shows (s,s) ∈ wp (x´= f & G) ⌈I⌉
  using assms unfolding wp-g-orbital diff-invariant-eq apply clarsimp
  by (erule-tac x=s in allE, frule ivp-solsD(2), clarsimp)

lemma diff-inv-axiom2:
  assumes picard-lindeloef (λt. f) UNIV UNIV 0
    and ⋀s. {t::real. t ≥ 0} ⊆ picard-lindeloef.ex-ivl (λt. f) UNIV UNIV 0 s
    and diff-invariant I (λt. f) (λs. {t::real. t ≥ 0}) UNIV 0 G
  shows wp (x´= f & G) ⌈I⌉ = wp ⌈G⌉ ⌈I⌉
proof(unfold wp-g-orbital, subst wp-rel, clarsimp simp: fun-eq-iff)
  fix s
  let ?ex-ivl s = picard-lindeloef.ex-ivl (λt. f) UNIV UNIV 0 s
  let ?lhs s =
    ∀ X∈Sols (λt. f) (λs. {t. t ≥ 0}) UNIV 0 s. ∀ t≥0. (∀ τ. 0 ≤ τ ∧ τ ≤ t ⟶ G (X τ)) ⟶ I (X t)
  obtain X where xivp1: X ∈ Sols (λt. f) (λs. ?ex-ivl s) UNIV 0 s
    using picard-lindeloef.flow-in-ivp-sols-ex-ivl[OF assms(1)] by auto
  have xivp2: X ∈ Sols (λt. f) (λs. Collect ((≤) 0)) UNIV 0 s
    by (rule in-ivp-sols-subset[OF - - xivp1], simp-all add: assms(2))
  hence shyp: X 0 = s
```

using *ivp-solsD* **by** *auto*
have *dinv*: $\forall s.\ I\ s \longrightarrow ?lhs\ s$
using *assms*(3) **unfolding** *diff-invariant-eq* **by** *auto*
{**assume** *?lhs s* **and** *G s*
hence *I s*
by (*erule-tac x=X* **in** *ballE*, *erule-tac x=0* **in** *allE*, *auto simp*: *shyp xivp2*)}
hence $?lhs\ s \longrightarrow (G\ s \longrightarrow I\ s)$
by *blast*
moreover
{**assume** $G\ s \longrightarrow I\ s$
hence *?lhs s*
apply(*clarify*, *subgoal-tac* $\forall\tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$)
apply(*erule-tac x=0* **in** *allE*, *frule ivp-solsD*(2), *simp*)
using *dinv* **by** *blast*+}
ultimately show $?lhs\ s = (G\ s \longrightarrow I\ s)$
by *blast*
qed

lemma *diff-inv-rule*:
assumes $\lceil P\rceil \leq \lceil I\rceil$ **and** *diff-invariant* $I\ f\ U\ S\ t_0\ G$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
shows $\lceil P\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$
apply(*rule wp-g-orbital-inv*[*OF assms*(1) - *assms*(3)])
unfolding *wp-diff-inv* **using** *assms*(2) **.**

end

0.6.6 Examples

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

theory *HS-VC-MKA-Examples-rel*
imports *HS-VC-MKA-rel*

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* :: $real\hat{}2 \Rightarrow real\hat{}2\ (f)$
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*:
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil \leq wp\ (x' = f\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*simp add*: *local-flow.wp-g-ode-subset*[*OF local-flow-pend*])

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* $\Rightarrow$ *real*ˆ*2* $\Rightarrow$ *real*ˆ*2* (f)
where $f\ g\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ g)$

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

— 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 $(x{::}real) \leq h$
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

lemma *bouncing-ball-inv*:
fixes *h*::*real*
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow \lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil \leq$
wp
(*LOOP*
$((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)$
$)\ \lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$

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*: $g * \tau^2 \;/\; 2 + v * \tau + (x::real) = 0$
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*: *monoid-mult-class.power2-eq-square*)
qed

lemma *inv-conserv-at-air*[*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
shows $2 * g * (g * \tau^2 \;/\; 2 + v * \tau + (x::real)) =$
$2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
have $?lhs = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
also have $... = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** $?rhs = g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
also have ... = *?middle*
by (*simp add*: *semiring-normalization-rules*(*29*))
finally have *?rhs* = *?middle*.}
ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\; s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$
$(LOOP$
$((EVOL\ (\varphi\ g)\ (\lambda s.\ 0 \leq s\$1)\ T);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
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 $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil \leq wp$
$(LOOP$
$((x' = (\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
apply(*rule wp-loopI*, *simp-all add*: *local-flow.wp-g-ode-subset*[*OF local-flow-ball*])
by (*auto 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 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* (φ a L)
  by (*unfold-locales*, *auto intro*!: *poly-derivatives local-lipschitz-temp-dyn simp*: *forall-4 vec-eq-iff*)

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

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

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

lemma *thermostat*:
  assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
  shows $\lceil\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0\rceil \leq wp$
  (*LOOP*
    — control
    $((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
    $(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$

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

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

end

0.7 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 *HS-VC-MKA-ndfun*
imports
../HS-ODEs HS-VC-MKA
Transformer-Semantics.Kleisli-Quantale
KAD.Modal-Kleene-Algebra

begin

0.7.1 Non-deterministic functions

Our semantics now corresponds to nondeterministic functions *'a nd-fun*. Below we prove some auxiliary lemmas for them and show that they form an antidomain kleene algebra. The proof just extends the results on the Transformer_Semantics.Kleisli_Quantale theory.

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

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*) *antidomain-kleene-algebra*
begin

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

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-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*)

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

0.7.2 Store and weakest preconditions

Now that we know that nondeterministic functions form an Antidomain Kleene Algebra, we give a lifting operation from predicates to *'a nd-fun* and use it to compute weakest liberal preconditions.

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

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

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
  and *Relation.relcomp* (**infixl** ; *75*)
  and *Range-Semiring.antirange-semiring-class.ars-r* (*r*)
  and *antidomain-semiringl.ads-d* (*d*)

abbreviation *p2ndf* :: *'a pred* ⇒ *'a nd-fun* ((*1*⌈-⌉))
  where ⌈*Q*⌉ ≡ (λ *x*::*'a*. {*s*::*'a*. *s* = *x* ∧ *Q s*})$^\bullet$

lemma *p2ndf-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*⌉
  ad ⌈*P*⌉ = ⌈λ*s*. ¬ *P s*⌉
  d ⌈*P*⌉ = ⌈*P*⌉ ⌈*P*⌉ ≤ η$^\bullet$
  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* ⌈*P*⌉ = ⌈λ*s*. ∀ *s'*. *s'* ∈ ((*F*$_\bullet$) *s*) ⟶ *P s'*⌉
  apply(*simp add*: *fbox-def antidomain-op-nd-fun-def*)
  by(*rule nd-fun-ext*, *auto simp*: *Rep-comp-hom kcomp-prop*)

definition *vec-upd* :: (*'a*ˆ*'b*) ⇒ *'b* ⇒ *'a* ⇒ *'a*ˆ*'b*
  where *vec-upd s i a* = (χ *j*. (((\$) *s*)(*i* := *a*)) *j*)

lemma *vec-upd-eq*: *vec-upd s i a* = (χ *j*. *if j* = *i then a else s*\$*j*)
  by (*simp add*: *vec-upd-def*)

definition *assign* :: *'b* ⇒ (*'a*ˆ*'b* ⇒ *'a*) ⇒ (*'a*ˆ*'b*) *nd-fun* ((*2*- ::= -) [*70*, *65*] *61*)
  where (*x* ::= *e*) = (λ*s*. {*vec-upd s x* (*e s*)})$^\bullet$

abbreviation *seq-comp* :: *'a nd-fun* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (**infixl** ; *75*)
  where *f* ; *g* ≡ *f* · *g*

lemma *wp-assign*[*simp*]: *wp* (*x* ::= *e*) ⌈*Q*⌉ = ⌈λ*s*. *Q* (χ *j*. (((\$) *s*)(*x* := (*e s*))) *j*)⌉
  unfolding *wp-nd-fun nd-fun-eq-iff vec-upd-def assign-def* **by** *auto*

definition *nondet-assign* :: *'b* ⇒ (*'a*ˆ*'b*) *nd-fun* ((*2*- ::= *?*) [*70*] *61*)
  where (*x* ::= *?*) = (λ*s*. {(*vec-upd s x k*)|*k*. *True*})$^\bullet$

lemma *wp-nondet-assign*[*simp*]: *wp* (*x* ::= *?*) ⌈*P*⌉ = ⌈λ*s*. ∀ *k*. *P* (χ *j*. (((\$) *s*)(*x* := *k*)) *j*)⌉
  unfolding *wp-nd-fun nondet-assign-def vec-upd-eq* **apply**(*clarsimp*, *safe*)
  by (*erule-tac x*=(χ *j*. *if j* = *x then k else s* \$ *j*) **in** *allE*, *auto*)

abbreviation *skip* :: *'a nd-fun*
  where *skip* ≡ *1*

abbreviation *cond-sugar* :: *'a pred* ⇒ *'a nd-fun* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (*IF* - *THEN* - *ELSE* - [*64*,*64*] *63*)
  where *IF P THEN X ELSE Y* ≡ *aka-cond* ⌈*P*⌉ *X Y*

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

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

0.7.3 Verification of hybrid programs

Verification by providing evolution

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

lemma $wp\text{-}g\text{-}dyn[simp]$:
  fixes φ :: $('a::preorder) \Rightarrow 'b \Rightarrow 'b$
  shows $wp\ (EVOL\ \varphi\ G\ U)\ \lceil Q\rceil = \lceil \lambda s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$
  unfolding *wp-nd-fun g-evol-def g-orbit-eq* **by** (*auto simp*: *fun-eq-iff*)

Verification by providing solutions

definition $g\text{-}ode$ ::$(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow$
  $real \Rightarrow 'a\ nd\text{-}fun\ ((1x' = \text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}))$
  where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0) \equiv (\lambda\ s.\ g\text{-}orbital\ f\ G\ U\ S\ t_0\ s)^{\bullet}$

lemma $wp\text{-}g\text{-}orbital$: $wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil =$
  $\lceil \lambda s.\ \forall X{\in}Sols\ f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)\rceil$
  unfolding *g-orbital-eq*(*1*) *wp-nd-fun g-ode-def* **by** (*auto simp*: *fun-eq-iff*)

context *local-flow*
begin

lemma *wp-g-ode-subset*:
  assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
  shows $wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \lceil Q\rceil =$
  $\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
  apply(*unfold wp-g-orbital*, *clarsimp*, *rule iffI*; *clarify*)
   apply(*force simp*: *in-ivp-sols assms*)
  apply(*frule ivp-solsD*(*2*), *frule ivp-solsD*(*3*), *frule ivp-solsD*(*4*))
  apply(*subgoal-tac* $\forall \tau{\in}down\ (U\ s)\ t.\ X\ \tau = \varphi\ \tau\ s$)
   apply(*clarsimp*, *fastforce*, *rule ballI*)
  apply(*rule ivp-unique-solution*[*OF - - - - - in-ivp-sols*])
  using *assms* **by** *auto*

lemma *wp-g-ode*: $wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ \lceil Q\rceil =$
  $\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
  by (*subst wp-g-ode-subset*, *simp-all add*: *init-time interval-time*)

lemma *wp-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ \lceil Q\rceil =$
  $\lceil \lambda s.\ s \in S \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
  apply(*subst wp-g-ode-subset*, *simp-all add*: *subintervalI init-time real-Icc-closed-segment*)
  by (*auto simp*: *closed-segment-eq-real-ivl*)

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 $g\text{-}ode\text{-}inv$ :: $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow$
  $real \Rightarrow 'a\ pred \Rightarrow 'a\ nd\text{-}fun\ ((1x'{=}\text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}\ DINV\ \text{-}\ ))$
  where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$

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

lemma *wp-g-orbital-inv*:
  assumes $\lceil P\rceil \leq \lceil I\rceil$ **and** $\lceil I\rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
  shows $\lceil P\rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$

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

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

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

context *local-flow*
begin

lemma *wp-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
shows $diff\text{-}invariant\ I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$
$(\lceil\lambda s.\ s \in S \longrightarrow I\ s\rceil = wp\ (x\acute{}= (\lambda t.\ f)\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ 0)\ \lceil\lambda s.\ s \in S \longrightarrow I\ s\rceil)$
unfolding *wp-diff-inv[symmetric]*
apply(*subst wp-g-ode-subset[OF assms]*, *simp*)+
apply(*clarsimp*, *safe*, *force*)
apply(*erule-tac x=0* **in** *ballE*)
using *init-time in-domain ivp(2) assms* **apply**(*force*, *force*)
apply(*erule-tac x=s* **in** *allE*, *clarsimp*, *erule-tac x=t* **in** *ballE*)
using *in-domain ivp(2) assms* **by** *force*+

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

end

lemma *wp-g-odei*: $\lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil \Longrightarrow \lceil\lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$\lceil P\rceil \leq wp\ (x\acute{}= f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I)\ \lceil Q\rceil$
unfolding *g-ode-inv-def*
apply(*rule-tac b=wp* $(x\acute{}= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil$ **in** *order.trans*)
apply(*rule-tac I=I* **in** *wp-g-orbital-inv*, *simp-all*)
apply(*subst wp-g-orbital-guard*, *simp*)
by (*rule fbox-iso*, *simp*)

0.7.4 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.

abbreviation *g-dl-ode* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a nd-fun ((1x´=- & -))
where $(x\acute{}= f\ \&\ G) \equiv (x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

abbreviation *g-dl-ode-inv* :: (('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a nd-fun ((1x´=- & - DINV -))
where $(x\acute{}= f\ \&\ G\ DINV\ I) \equiv (x\acute{}=(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0\ DINV\ I)$

lemma *diff-solve-axiom1*:
assumes *local-flow f UNIV UNIV* φ
shows $wp\ (x\acute{}= f\ \&\ G)\ \lceil Q\rceil =$
$\lceil\lambda s.\ \forall t{\geq}0.\ (\forall\tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$
by (*subst local-flow.wp-g-ode-subset[OF assms]*, *auto*)

lemma *diff-solve-axiom2*:
fixes c::$'a$::{*heine-borel*, *banach*}
shows *wp* ($x´$=($\lambda s.\ c$) & G) $\lceil Q \rceil$ =
$\lceil \lambda s.\ \forall t \geq 0.\ (\forall \tau \in \{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow Q\ (s + t *_R c) \rceil$
apply(*subst local-flow.wp-g-ode-subset*[**where** φ=($\lambda t\ s.\ s + t *_R c$) **and** T=*UNIV*])
by (*rule line-is-local-flow*, *auto*)

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t \geq 0.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P \rceil \leq$ *wp* ($x´$= f & G) $\lceil Q \rceil$
using *assms* **by** (*subst local-flow.wp-g-ode-subset*[*OF assms*(*1*)], *auto*)

lemma *diff-weak-axiom1*: $\eta^\bullet \leq$ *wp* ($x´$= f & G *on* $U\ S$ @ t_0) $\lceil G \rceil$
unfolding *wp-nd-fun g-ode-def g-orbital-eq less-eq-nd-fun-def*
by (*auto simp*: *le-fun-def*)

lemma *diff-weak-axiom2*:
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*:
assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq$ *wp* ($x´$= f & G *on* $U\ S$ @ t_0) $\lceil Q \rceil$
using *assms* **by** (*auto simp*: *wp-g-orbital*)

lemma *wp-g-orbit-IdD*:
assumes *wp* ($x´$= f & G *on* $U\ S$ @ t_0) $\lceil C \rceil = \eta^\bullet$
and $\forall \tau \in$(*down* ($U\ s$) t). $x\ \tau \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
shows $\forall \tau \in$(*down* ($U\ s$) t). C ($x\ \tau$)
proof
fix τ **assume** $\tau \in$ (*down* ($U\ s$) t)
hence $x\ \tau \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
using *assms*(*2*) **by** *blast*
also have $\forall y.\ y \in$ (*g-orbital* $f\ G\ U\ S\ t_0\ s$) $\longrightarrow C\ y$
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 *wp* ($x´$= f & G *on* $U\ S$ @ t_0) $\lceil C \rceil = \eta^\bullet$
shows *wp* ($x´$= f & G *on* $U\ S$ @ t_0) $\lceil Q \rceil$ = *wp* ($x´$= f & ($\lambda s.\ G\ s \wedge C\ s$) *on* $U\ S$ @ 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* $U\ S$ @ t_0)$_\bullet$) $s \subseteq$ (($x´$= f & ($\lambda s.\ G\ s \wedge C\ s$) *on* $U\ S$ @ t_0)$_\bullet$) s
proof(*clarsimp simp*: *g-ode-def*)
fix s' **assume** $s' \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in$ *ivp-sols* $f\ U\ S\ t_0\ s$
and $X\ \tau = s'$ **and** $\tau \in U\ s$ **and** *guard-x*:($\mathcal{P}\ X$ (*down* ($U\ s$) τ) $\subseteq \{s.\ G\ s\}$)
using *g-orbitalD*[*of* $s'\ f\ G$ - $S\ t_0\ s$] **by** *blast*
have $\forall t \in$(*down* ($U\ s$) τ). $\mathcal{P}\ X$ (*down* ($U\ s$) t) $\subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall t \in$(*down* ($U\ s$) τ). $t \in$ ($U\ s$)
using ‹$\tau \in$ ($U\ s$)› **by** *auto*
ultimately have $\forall t \in$(*down* ($U\ s$) τ). $X\ t \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence $\forall t \in$(*down* ($U\ s$) τ). C ($X\ t$)
using *wp-g-orbit-IdD*[*OF assms*(*1*)] **by** *blast*

thus $s' \in$ *g-orbital f* $(\lambda s.\ G\ s \wedge C\ s)\ U\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp* ‹$\tau \in (U\ s)$›] *guard-x* ‹$X\ \tau = s'$› **by** *fastforce*
qed
next
fix s **show** $((x\acute{} = f\ \&\ \lambda s.\ G\ s \wedge C\ s\ on\ U\ S\ @\ t_0)_\bullet)\ s \subseteq ((x\acute{} = f\ \&\ G\ on\ U\ S\ @\ t_0)_\bullet)\ s$
by (*auto simp*: *g-orbital-eq g-ode-def*)
qed

lemma *diff-cut-rule*:
assumes *wp-C*: $\lceil P \rceil \leq wp\ (x\acute{} = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil$
and *wp-Q*: $\lceil P \rceil \leq wp\ (x\acute{} = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x\acute{} = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
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** $t \in U\ s$
and *x-ivp*:$X \in$ *ivp-sols* $f\ U\ S\ t_0\ s$
and *guard-x*:$\forall x.\ x \in U\ s \wedge x \leq t \longrightarrow G\ (X\ x)$
have $\forall t \in (down\ (U\ s)\ t).\ X\ t \in$ *g-orbital* $f\ G\ U\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
hence $\forall t \in (down\ (U\ s)\ t).\ C\ (X\ t)$
using *wp-C* ‹$P\ s$› **by** (*subst* (*asm*) *wp-nd-fun*, *auto simp*: *g-ode-def*)
hence $X\ t \in$ *g-orbital* $f\ (\lambda s.\ G\ s \wedge C\ s)\ U\ S\ t_0\ s$
using *guard-x* ‹$t \in (U\ s)$› **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

lemma *diff-inv-axiom1*:
assumes $G\ s \longrightarrow I\ s$ **and** *diff-invariant* $I\ (\lambda t.\ f)\ (\lambda s.\ \{t.\ t \geq 0\})$ *UNIV 0* G
shows $s \in ((wp\ (x\acute{} = f\ \&\ G)\ \lceil I \rceil)_\bullet)\ s$
using *assms* **unfolding** *wp-g-orbital diff-invariant-eq* **apply** *clarsimp*
by (*erule-tac x=s* **in** *allE*, *frule ivp-solsD*(*2*), *clarsimp*)

lemma *diff-inv-axiom2*:
assumes *picard-lindeloef* $(\lambda t.\ f)$ *UNIV UNIV 0*
and $\bigwedge s.\ \{t::real.\ t \geq 0\} \subseteq$ *picard-lindeloef.ex-ivl* $(\lambda t.\ f)$ *UNIV UNIV 0 s*
and *diff-invariant* $I\ (\lambda t.\ f)\ (\lambda s.\ \{t::real.\ t \geq 0\})$ *UNIV 0* G
shows $wp\ (x\acute{} = f\ \&\ G)\ \lceil I \rceil = wp\ \lceil G \rceil\ \lceil I \rceil$
proof(*unfold wp-g-orbital*, *subst wp-nd-fun*, *clarsimp simp*: *fun-eq-iff*)
fix s
let *?ex-ivl s* = *picard-lindeloef.ex-ivl* $(\lambda t.\ f)$ *UNIV UNIV 0 s*
let *?lhs s* =
$\forall X \in Sols\ (\lambda t.\ f)\ (\lambda s.\ \{t.\ t \geq 0\})$ *UNIV 0 s*. $\forall t \geq 0.\ (\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)) \longrightarrow I\ (X\ t)$
obtain X **where** *xivp1*: $X \in Sols\ (\lambda t.\ f)\ (\lambda s.$ *?ex-ivl s*) *UNIV 0 s*
using *picard-lindeloef.flow-in-ivp-sols-ex-ivl*[*OF assms*(*1*)] **by** *auto*
have *xivp2*: $X \in Sols\ (\lambda t.\ f)\ (\lambda s.\ Collect\ ((\leq)\ 0))$ *UNIV 0 s*
by (*rule in-ivp-sols-subset*[*OF - - xivp1*], *simp-all add*: *assms*(*2*))
hence *shyp*: $X\ 0 = s$
using *ivp-solsD* **by** *auto*
have *dinv*: $\forall s.\ I\ s \longrightarrow$ *?lhs s*
using *assms*(*3*) **unfolding** *diff-invariant-eq* **by** *auto*
{**assume** *?lhs s* **and** $G\ s$
hence $I\ s$
by (*erule-tac x=X* **in** *ballE*, *erule-tac x=0* **in** *allE*, *auto simp*: *shyp xivp2*)}
hence *?lhs s* $\longrightarrow (G\ s \longrightarrow I\ s)$
by *blast*
moreover
{**assume** $G\ s \longrightarrow I\ s$
hence *?lhs s*
apply(*clarify*, *subgoal-tac* $\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$)
apply(*erule-tac x=0* **in** *allE*, *frule ivp-solsD*(*2*), *simp*)

using *dinv* **by** *blast+*}
ultimately show *?lhs s* $= (G\ s \longrightarrow I\ s)$
by *blast*
qed

lemma *diff-inv-rule*:
assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** *diff-invariant* $I\ f\ U\ S\ t_0\ G$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
apply(*rule wp-g-orbital-inv*[*OF assms*(*1*) - *assms*(*3*)])
unfolding *wp-diff-inv* **using** *assms*(*2*) **.**

end

0.7.5 Examples

We prove partial correctness specifications of some hybrid systems with our recently described verification components. Notice that this is an exact copy of the file *HS-VC-MKA-Examples*, meaning our components are truly modular and we can choose either a relational or predicate transformer semantics.

theory *HS-VC-MKA-Examples-ndfun*
imports *HS-VC-MKA-ndfun*

begin

Pendulum

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

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

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

— Verified 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*:
$\lceil \lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp\ (x' = f\ \&\ G)\ \lceil \lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
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*:
$\lceil \lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp\ (x' = f\ \&\ G)\ \lceil \lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2 \rceil$

by (*simp add*: *local-flow.wp-g-ode-subset*[*OF local-flow-pend*])

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* :: $real \Rightarrow real\hat{\ }2 \Rightarrow real\hat{\ }2$ (f)
where $f\ g\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ g)$

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

— 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 $(x{::}real) \leq h$
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

lemma *bouncing-ball-inv*:
fixes *h*::*real*
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow \lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil \leq$
wp
(*LOOP*
$((x\acute{} = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s. - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)$
$)\ \lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
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*: $g * \tau^2\ /\ 2 + v * \tau + (x{::}real) = 0$
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*: *monoid-mult-class.power2-eq-square*)
qed

lemma *inv-conserv-at-air*[*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
shows $2 * g * (g * \tau^2 \ / \ 2 + v * \tau + (x{::}real)) =$
$2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
have *?lhs* $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$
by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
also have ... $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** *?rhs* $= g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$
by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
also have ... = *?middle*
by (*simp add*: *semiring-normalization-rules*(*29*))
finally have *?rhs* = *?middle*.}
ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil\lambda s.\ s\$1 = h \wedge s\$2 = 0\rceil \leq wp$
(*LOOP*
((*EVOL* $(\varphi\ g)$ $(\lambda s.\ 0 \leq s\$1)$ *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* $(\varphi\ g)$
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*:
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*
(($x' = (\lambda t.\ f\ g)$ & $(\lambda\ s.\ s\$1 \geq 0)$ *on* $(\lambda s.\ UNIV)$ *UNIV* @ *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-subset*[*OF local-flow-ball*])
by (*auto 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 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 $(L -\ Tmax)\ /\ (L -\ T) \leq exp\ (-a * t) \wedge exp\ (-a * t) \leq 1$
    using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
  moreover have $L -\ T > 0$
    using *Thyps* **by** *auto*
  ultimately have *obs*: $(L -\ Tmax) \leq exp\ (-a * t) * (L -\ T) \wedge exp\ (-a * t) * (L -\ T) \leq (L -\ T)$
    by (*simp add*: *pos-divide-le-eq*)
  thus $(L -\ Tmax) \leq exp\ (-(a * t)) * (L -\ T)$
    by *auto*
  thus $L -\ exp\ (-(a * t)) * (L -\ T) \leq Tmax$
    by *auto*
  show $Tmin \leq L -\ exp\ (-(a * t)) * (L -\ T)$
    using *Thyps* **and** *obs* **by** *auto*
qed

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

lemma *thermostat*:
  assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
  shows $\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$
  (*LOOP*
    — control
    $((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
    $(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
    $(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax -\ 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
    — dynamics
    $(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x\acute{}=(\lambda t.\ f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV$
$@\ 0)$
    $ELSE\ (x\acute{}=(\lambda t.\ f\ a\ L)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ ((L-Tmax)/(L-s\$3)))/a)\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0))\ )$
  $INV\ (\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
  $\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \rceil$
  apply(*rule wp-loopI*, *simp-all add*: *fbox-temp-dyn*[*OF assms*(*1*,*2*)])
  using *temp-dyn-up-real-arith*[*OF assms*(*1*) - - *assms*(*4*), *of Tmin*]
    and *temp-dyn-down-real-arith*[*OF assms*(*1*,*3*), *of - Tmax*] **by** *auto*

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

end

0.8 Verification components with KAT

We create verification rules based on various Kleene Algebras.

theory *HS-VC-KAT*
 imports *KAT-and-DRA.PHL-KAT*

begin

0.8.1 Hoare logic in KAT

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

notation t ($\mathtt{tt}$)

hide-const t

no-notation *if-then-else* (*if - then - else - fi* [*64*,*64*,*64*] *63*)
 and *HOL.If* ((*if* (-)/ *then* (-)/ *else* (-)) [*0*, *0*, *10*] *10*)
 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**
 $H\ p\ x\ q \longleftrightarrow \mathtt{tt}\ p \cdot x \leq x \cdot \mathtt{tt}\ q$

lemma *H-consl*: $\mathtt{tt}\ p \leq \mathtt{tt}\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$
 using *Hoare-def phl-cons1* **by** *blast*

lemma *H-consr*: $\mathtt{tt}\ q' \leq \mathtt{tt}\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$
 using *Hoare-def phl-cons2* **by** *blast*

lemma *H-cons*: $\mathtt{tt}\ p \leq \mathtt{tt}\ p' \Longrightarrow \mathtt{tt}\ q' \leq \mathtt{tt}\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$
 by (*simp add*: *H-consl H-consr*)

— Skip

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

— Abort

lemma *H-abort*: $H\ p\ 0\ q$
 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*)

— Nondeterministic choice

lemma *H-choice*: $H\ p\ x\ q \Longrightarrow H\ p\ y\ q \Longrightarrow H\ p\ (x + y)\ q$
  using *local.distrib-left local.join.sup.mono* **by** (*auto simp*: *Hoare-def*)

— Conditional statement

definition *kat-cond* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*if - then - else -* [*64*,*64*,*64*] *63*) **where**
  if p then x else y = ($\mathfrak{tt}\ p \cdot x + n\ p \cdot y$)

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

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

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

— While loop

definition *kat-while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do -* [*64*,*64*] *63*) **where**
  while b do x = $(\mathfrak{tt}\ b \cdot x)^\star \cdot n\ b$

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

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

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

lemma *H-while-inv*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i \Longrightarrow \mathfrak{tt}\ i \cdot n\ r \leq \mathfrak{tt}\ q \Longrightarrow H\ (\mathfrak{tt}\ i \cdot \mathfrak{tt}\ r)\ x\ i \Longrightarrow H\ p$ (*while r inv i do x*) 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 $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i$ **and** $H\ i\ x\ i$ **and** $(\mathfrak{tt}\ i) \leq (\mathfrak{tt}\ q)$
  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*: $\mathtt{tt}\ p \leq \mathtt{tt}\ i \Longrightarrow H\ i\ x\ i \Longrightarrow \mathtt{tt}\ i \leq \mathtt{tt}\ q \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ q$
  unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

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

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

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

end

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

— Skip

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

— Abort

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

— Nondeterministic choice

lemma *R-choice*: (*Ref p q*) + (*Ref p q*) $\leq$ *Ref p q*
  unfolding *spec-def* **by** (*rule H-choice*) (*rule R1*)+

— Conditional statement

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

— While loop

lemma *R-while*: *while q do* (*Ref* (tt *p* $\cdot$ tt *q*) *p*) $\leq$ *Ref p* (tt *p* $\cdot$ *n q*)
proof –
  have *H* (tt *p* $\cdot$ tt *q*) (*Ref* (tt *p* $\cdot$ tt *q*) *p*) *p*
    by (*simp-all add*: *R1*)
  hence *H p* (*while q do* (*Ref* (tt *p* $\cdot$ tt *q*) *p*)) (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*: $\mathtt{tt}\ p \leq \mathtt{tt}\ i \Longrightarrow \mathtt{tt}\ i \leq \mathtt{tt}\ q \Longrightarrow Ref\ i\ i \leq Ref\ p\ q$
using *R-cons* **by** *force*

end

end

0.9 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 *HS-VC-KAT-rel*
imports
HS-VC-KAT
../HS-ODEs

begin

— We start by deleting some conflicting notation.

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
and *Archimedean-Field.floor-ceiling-class.floor* (⌊-⌋)
and *tau* (τ)
and *n-op* (*n* - [*90*] *91*)

notation *Id* (*skip*)
and *relcomp* (**infixl** ; *70*)

0.9.1 Relational model

context *dioid-one-zero*
begin

lemma *power-inductl*: $z + x \cdot y \leq y \Longrightarrow (x\ \hat{}\ n) \cdot z \leq y$
by(*induct n*, *auto*, *metis mult.assoc mult-isol order-trans*)

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 ‹$z + y \cdot x \leq y$› **by** *auto*
    ultimately have $z \cdot x \mathbin{\hat{}} \mathit{Suc}\ n \leq y$ **by** *auto*}
  thus *?case*
    by (*metis Suc*)
qed

end

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

lemma *power-is-relpow*: *rel-dioid.power* $X\ n = X \mathbin{\hat{}\hat{}} n$
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*: $X\hat{}* = (\bigcup n.\ \mathit{rel\text{-}dioid.power}\ X\ n)$
  by (*simp add*: *power-is-relpow rtrancl-is-UN-relpow*)

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

lemma *rel-star-contr*: $X\hat{}*\ O\ Y = (\bigcup n.\ (\mathit{rel\text{-}dioid.power}\ X\ n)\ O\ Y)$
  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-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

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

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

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

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

lemma *RdL-is-rRKAT*: (∀ *x*. {(*x*,*x*)}; *R1* ⊆ {(*x*,*x*)}; *R2*) = (*R1* ⊆ *R2*)
by *auto*

0.9.2 Store and Hoare triples

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

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^'b*) ⇒ *'b* ⇒ *'a* ⇒ *'a^'b*
where *vec-upd s i a* ≡ (χ *j*. (((\$) *s*)(*i* := *a*)) *j*)

lemma *vec-upd-eq*: *vec-upd s i a* = (χ *j*. *if j* = *i then a else s\$j*)
by (*simp add*: *vec-upd-def*)

definition *assign* :: *'b* ⇒ (*'a^'b* ⇒ *'a*) ⇒ (*'a^'b*) *rel* ((*2*- ::= -) [*70*, *65*] *61*)
where (*x* ::= *e*) ≡ {(*s*, *vec-upd s x* (*e s*))| *s*. *True*}

lemma *H-assign*: *P* = (λ*s*. *Q* (χ *j*. (((\$) *s*)(*x* := (*e s*))) *j*)) ⟹ *rel-kat.Hoare* ⌈*P*⌉ (*x* ::= *e*) ⌈*Q*⌉

unfolding *rel-kat-H assign-def vec-upd-def* **by** *force*

lemma *sH-assign*[*simp*]: *rel-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*)

definition *nondet-assign* :: $'b \Rightarrow ('a\hat{\ }'b)$ *rel* $((2\text{-} ::= ?\ )\ [70]\ 61)$
where $(x ::= ?) = \{(s, \textit{vec-upd}\ s\ x\ k)|s\ k.\ \textit{True}\}$

lemma *wp-nondet-assign*[*simp*]: *rel-kat.Hoare* $\lceil \lambda s.\ \forall k.\ P\ (\chi\ j.\ ((\$)\ s)(x := k))\ j) \rceil$ $(x ::= ?)$ $\lceil P \rceil$
unfolding *rel-kat-H nondet-assign-def vec-upd-eq* **apply** *clarsimp*
by (*erule-tac* $x{=}k$ **in** *allE*, *auto simp*: *fun-upd-def*)

— Next, the Hoare rule of the composition

lemma *H-seq*: *rel-kat.Hoare* $\lceil P \rceil$ X $\lceil R \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil R \rceil$ Y $\lceil Q \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ $(X\ ;\ Y)$ $\lceil Q \rceil$
by (*auto intro*: *rel-kat.H-seq*)

lemma *sH-seq*:
rel-kat.Hoare $\lceil P \rceil$ $(X\ ;\ Y)$ $\lceil Q \rceil =$ *rel-kat.Hoare* $\lceil P \rceil$ (X) $\lceil \lambda s.\ \forall s'.\ (s,\ s') \in Y \longrightarrow Q\ s' \rceil$
unfolding *rel-kat-H* **by** *auto*

— Rewriting the Hoare rule for the conditional statement

abbreviation *cond-sugar* :: $'a$ *pred* $\Rightarrow$ $'a$ *rel* $\Rightarrow$ $'a$ *rel* $\Rightarrow$ $'a$ *rel* (*IF - THEN - ELSE -* [*64*,*64*] *63*)
where *IF B THEN X ELSE Y* $\equiv$ *rel-kat.kat-cond* $\lceil B \rceil$ X Y

lemma *H-cond*: *rel-kat.Hoare* $\lceil \lambda s.\ P\ s \wedge B\ s \rceil$ X $\lceil Q \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil \lambda s.\ P\ s \wedge \neg\ B\ s \rceil$ Y $\lceil Q \rceil \Longrightarrow$
rel-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-kat.Hoare* $\lceil P \rceil$ (*IF B THEN X ELSE Y*) $\lceil Q \rceil \longleftrightarrow$
(*rel-kat.Hoare* $\lceil \lambda s.\ P\ s \wedge B\ s \rceil$ X $\lceil Q \rceil \wedge$ *rel-kat.Hoare* $\lceil \lambda s.\ P\ s \wedge \neg\ B\ s \rceil$ Y $\lceil Q \rceil$)
by (*auto simp*: *rel-kat.H-cond-iff rel-kat-H*)

— Rewriting the Hoare rule for the while loop

abbreviation *while-inv-sugar* :: $'a$ *pred* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *rel* $\Rightarrow$ $'a$ *rel* (*WHILE - INV - DO -* [*64*,*64*,*64*] *63*)

where *WHILE B INV I DO X* $\equiv$ *rel-kat.kat-while-inv* $\lceil B \rceil$ $\lceil I \rceil$ X

lemma *sH-while-inv*: $\forall s.\ P\ s \longrightarrow I\ s \Longrightarrow \forall s.\ I\ s \wedge \neg\ B\ s \longrightarrow Q\ s \Longrightarrow$ *rel-kat.Hoare* $\lceil \lambda s.\ I\ s \wedge B\ s \rceil$ X $\lceil I \rceil$
$\Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ (*WHILE B INV I DO X*) $\lceil Q \rceil$
by (*rule rel-kat.H-while-inv*, *auto simp*: *p2r-def rel-kat.Hoare-def*, *fastforce*)

— Finally, we add a Hoare triple rule for finite iterations.

abbreviation *loopi-sugar* :: $'a$ *rel* $\Rightarrow$ $'a$ *pred* $\Rightarrow$ $'a$ *rel* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP X INV I* $\equiv$ *rel-kat.kat-loop-inv* X $\lceil I \rceil$

lemma *H-loop*: *rel-kat.Hoare* $\lceil P \rceil$ X $\lceil P \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ (*LOOP X INV I*) $\lceil P \rceil$
by (*auto intro*: *rel-kat.H-loop*)

lemma *H-loopI*: *rel-kat.Hoare* $\lceil I \rceil$ X $\lceil I \rceil \Longrightarrow \lceil P \rceil \subseteq \lceil I \rceil \Longrightarrow \lceil I \rceil \subseteq \lceil Q \rceil \Longrightarrow$ *rel-kat.Hoare* $\lceil P \rceil$ (*LOOP X INV I*) $\lceil Q \rceil$
using *rel-kat.H-loop-inv*[*of* $\lceil P \rceil$ $\lceil I \rceil$ X $\lceil Q \rceil$] **by** *auto*

0.9.3 Verification of hybrid programs

— Verification by providing evolution

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

lemma *sH-g-evol*[*simp*]:
  fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
  shows $rel\text{-}kat.Hoare\ \lceil P\rceil\ (EVOL\ \varphi\ G\ U)\ \lceil Q\rceil = (\forall s.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
  unfolding *rel-kat-H g-evol-def g-orbit-eq* **by** *auto*

lemma *H-g-evol*:
  fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
  assumes $P = (\lambda s.\ (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
  shows $rel\text{-}kat.Hoare\ \lceil P\rceil\ (EVOL\ \varphi\ G\ U)\ \lceil Q\rceil$
  by (*simp add*: *assms*)

— Verification by providing solutions

definition *g-ode* :: $(real \Rightarrow ('a::banach){\Rightarrow}'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow real \Rightarrow$
  $'a\ rel\ ((1x\acute{}{=}\text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}))$
  where $(x\acute{}{=}\ f\ \&\ G\ on\ T\ S\ @\ t_0) = \{(s,s')\ |s\ s'.\ s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s\}$

lemma *H-g-orbital*:
  $P = (\lambda s.\ (\forall X{\in}ivp\text{-}sols\ f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \Longrightarrow$
  $rel\text{-}kat.Hoare\ \lceil P\rceil\ (x\acute{}{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$
  unfolding *rel-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

lemma *sH-g-orbital*: $rel\text{-}kat.Hoare\ \lceil P\rceil\ (x\acute{}{=}\ f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil =$
  $(\forall s.\ P\ s \longrightarrow (\forall X{\in}ivp\text{-}sols\ f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)))$
  unfolding *g-orbital-eq g-ode-def rel-kat-H* **by** *auto*

context *local-flow*
begin

lemma *sH-g-ode-subset*:
  assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
  shows $rel\text{-}kat.Hoare\ \lceil P\rceil\ (x\acute{}{=}\ (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \lceil Q\rceil =$
  $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
  fix $s\ t$
  assume *hyps*: $s \in S\ P\ s\ t \in U\ s\ \forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)$
    and *main*: $\forall s.\ P\ s \longrightarrow (\forall X{\in}Sols\ (\lambda t.\ f)\ U\ S\ 0\ s.\ \forall t{\in}U\ s.\ (\forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
  hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ U\ S\ 0\ s$
    using *in-ivp-sols assms* **by** *blast*
  thus $Q\ (\varphi\ t\ s)$
    using *main hyps* **by** *fastforce*
next
  fix $s\ X\ t$
  assume *hyps*: $P\ s\ X \in Sols\ (\lambda t.\ f)\ U\ S\ 0\ s\ t \in U\ s\ \ \forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (X\ \tau)$
    and *main*: $\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
  hence *obs*: $s \in S$
    using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
  hence $\forall \tau{\in}down\ (U\ s)\ t.\ X\ \tau = \varphi\ \tau\ s$
    using *eq-solution hyps assms* **by** *blast*
  thus $Q\ (X\ t)$
    using *hyps main obs* **by** *auto*

qed

lemma *H-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ s \in S \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}\textit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows $\textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \lceil Q\rceil$
using *assms* **apply**(*subst sH-g-ode-subset*[*OF assms*(*1*)])
unfolding *assms* **by** *auto*

lemma *sH-g-ode*: $\textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
by (*subst sH-g-ode-subset*, *auto simp*: *init-time interval-time*)

lemma *sH-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow \textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ \lceil Q\rceil$
$=$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
apply(*subst sH-g-ode-subset*; *clarsimp*, (*force*)*?*)
using *init-time interval-time mem-is-interval-1-I* **by** *blast*

lemma *sH-orbit*:
$\textit{rel-kat.Hoare}\ \lceil P\rceil\ (\{(s,s') \mid s\ s'.\ s' \in \gamma^{\varphi}\ s\})\ \lceil Q\rceil = (\forall s \in S.\ P\ s \longrightarrow (\forall\ t \in T.\ Q\ (\varphi\ t\ s)))$
using *sH-g-ode* **unfolding** *orbit-def g-ode-def* **by** *auto*

end

— Verification with differential invariants

definition *g-ode-inv* :: $(\textit{real} \Rightarrow ('a{::}\textit{banach}){\Rightarrow}'a) \Rightarrow 'a\ \textit{pred} \Rightarrow ('a \Rightarrow \textit{real set}) \Rightarrow 'a\ \textit{set} \Rightarrow$
$\textit{real} \Rightarrow 'a\ \textit{pred} \Rightarrow 'a\ \textit{rel}$ ((*1x´=- & - on - - @ - DINV -*))
where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ \textit{DINV}\ I) = (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$

lemma *sH-g-orbital-guard*:
assumes $R = (\lambda s.\ G\ s \wedge Q\ s)$
shows $\textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil = \textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$
$\lceil R\rceil$
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** $\textit{rel-kat.Hoare}\ \lceil I\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
shows $\textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q\rceil$
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*]: $\textit{rel-kat.Hoare}\ \lceil I\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil = \textit{diff-invariant}\ I\ f\ U\ S\ t_0\ G$
unfolding *diff-invariant-eq rel-kat-H g-orbital-eq g-ode-def* **by** *auto*

lemma *H-g-ode-inv*: $\textit{rel-kat.Hoare}\ \lceil I\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow$
$\lceil \lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow \textit{rel-kat.Hoare}\ \lceil P\rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ \textit{DINV}\ I)\ \lceil Q\rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $q'{=}\lceil \lambda s.\ I\ s \wedge G\ s\rceil$ **in** *rel-kat.H-consr*, *simp*)
apply(*subst sH-g-orbital-guard*[*symmetric*], *force*)
by (*rule-tac I=I* **in** *sH-g-orbital-inv*, *simp-all*)

0.9.4 Refinement Components

— Skip

lemma *R-skip*: $(\forall s.\ P\ s \longrightarrow Q\ s) \Longrightarrow \textit{Id} \leq \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
by (*simp add*: *rel-rkat.R2 rel-kat-H*)

— Composition

lemma *R-seq*: $(\textit{rel-R}\ \lceil P\rceil\ \lceil R\rceil)\ ;\ (\textit{rel-R}\ \lceil R\rceil\ \lceil Q\rceil) \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
using *rel-rkat.R-seq* **by** *blast*

lemma *R-seq-rule*: $X \le \textit{rel-R}\ \lceil P\rceil\ \lceil R\rceil \Longrightarrow Y \le \textit{rel-R}\ \lceil R\rceil\ \lceil Q\rceil \Longrightarrow X;\ Y \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-seq*)

lemmas *R-seq-mono* = *relcomp-mono*

— Assignment

lemma *R-assign*: $(x ::= e) \le \textit{rel-R}\ \lceil \lambda s.\ P\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j))\rceil\ \lceil P\rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-assign*, *clarsimp simp*: *fun-upd-def*)

lemma *R-assign-rule*: $(\forall\, s.\ P\ s \longrightarrow Q\ (\chi\ j.\ ((((\$)\ s)(x := (e\ s)))\ j))) \Longrightarrow (x ::= e) \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-assign*[*symmetric*] **by** (*rule rel-rkat.R2*)

lemma *R-assignl*: $P = (\lambda s.\ R\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j))) \Longrightarrow (x ::= e)\ ;\ \textit{rel-R}\ \lceil R\rceil\ \lceil Q\rceil \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*)
by (*rule-tac R-assign-rule*, *simp-all*)

lemma *R-assignr*: $R = (\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j))) \Longrightarrow \textit{rel-R}\ \lceil P\rceil\ \lceil R\rceil;\ (x ::= e) \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
apply(*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-assign-rule*, *simp*)

lemma $(x ::= e)\ ;\ \textit{rel-R}\ \lceil Q\rceil\ \lceil Q\rceil \le \textit{rel-R}\ \lceil(\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j)))\rceil\ \lceil Q\rceil$
by (*rule R-assignl*) *simp*

lemma $\textit{rel-R}\ \lceil Q\rceil\ \lceil(\lambda s.\ Q\ (\chi\ j.\ ((((\$)\ s)(x := e\ s))\ j)))\rceil;\ (x ::= e) \le \textit{rel-R}\ \lceil Q\rceil\ \lceil Q\rceil$
by (*rule R-assignr*) *simp*

— Conditional

lemma *R-cond*: $(\textit{IF}\ B\ \textit{THEN}\ \textit{rel-R}\ \lceil \lambda s.\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil\ \textit{ELSE}\ \textit{rel-R}\ \lceil \lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil) \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
using *rel-rkat.R-cond*[*of* $\lceil B\rceil\ \lceil P\rceil\ \lceil Q\rceil$] **by** *simp*

lemma *R-cond-mono*: $X \le X' \Longrightarrow Y \le Y' \Longrightarrow (\textit{IF}\ P\ \textit{THEN}\ X\ \textit{ELSE}\ Y) \le \textit{IF}\ P\ \textit{THEN}\ X'\ \textit{ELSE}\ Y'$
by (*auto simp*: *rel-kat.kat-cond-def*)

— While loop

lemma *R-while*: $\textit{WHILE}\ Q\ \textit{INV}\ I\ \textit{DO}\ (\textit{rel-R}\ \lceil \lambda s.\ P\ s \wedge Q\ s\rceil\ \lceil P\rceil) \le \textit{rel-R}\ \lceil P\rceil\ \lceil \lambda s.\ P\ s \wedge \neg\ Q\ s\rceil$
unfolding *rel-kat.kat-while-inv-def* **using** *rel-rkat.R-while*[*of* $\lceil Q\rceil\ \lceil P\rceil$] **by** *simp*

lemma *R-while-mono*: $X \le X' \Longrightarrow (\textit{WHILE}\ P\ \textit{INV}\ I\ \textit{DO}\ X) \subseteq \textit{WHILE}\ P\ \textit{INV}\ I\ \textit{DO}\ X'$
by (*simp add*: *rel-dioid.mult-isol rel-dioid.mult-isor rel-ka.conway.dagger-iso rel-kat.kat-while-def rel-kat.kat-while-inv-def*)

— Finite loop

lemma *R-loop*: $X \le \textit{rel-R}\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \le \lceil I\rceil \Longrightarrow \lceil I\rceil \le \lceil Q\rceil \Longrightarrow \textit{LOOP}\ X\ \textit{INV}\ I \le \textit{rel-R}\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *rel-rkat.spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \le X' \Longrightarrow \textit{LOOP}\ X\ \textit{INV}\ I \subseteq \textit{LOOP}\ X'\ \textit{INV}\ I$
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 $(EVOL\ \varphi\ G\ U) \leq$ *rel-R* $\lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)\rceil\ \lceil P\rceil$
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 $(\forall s.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (EVOL\ \varphi\ G\ U) \leq$ *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 $P = (\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(EVOL\ \varphi\ G\ U)$; *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 $R = (\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
rel-R $\lceil P\rceil\ \lceil R\rceil$; $(EVOL\ \varphi\ G\ U) \leq$ *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 $EVOL\ \varphi\ G\ U$; *rel-R* $\lceil Q\rceil\ \lceil Q\rceil \leq$ *rel-R* $\lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
by (*rule R-g-evoll*) *simp*

lemma
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b \Rightarrow 'b$
shows *rel-R* $\lceil Q\rceil\ \lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$; $EVOL\ \varphi\ G\ U \leq$ *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-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq$ *rel-R* $\lceil\lambda s.\ s{\in}S \longrightarrow (\forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-ode-subset*[*OF assms*], *simp-all*)

lemma *R-g-ode-rule-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
by (*rule rel-rkat.R2*, *subst sH-g-ode-subset*[*OF assms*], *auto*)

lemma *R-g-odel-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s))$
shows $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)$; *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*, *rule-tac R-g-ode-rule-subset*)
by (*simp-all add*: *assms*)

lemma *R-g-oder-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
and $R = (\lambda s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}\textit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows *rel-R* $\lceil P\rceil\ \lceil R\rceil$; $(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ U\ S\ @\ 0) \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
apply (*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-ode-rule-subset*, *simp-all add*: *assms*)

lemma *R-g-ode*:
$(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq$ *rel-R* $\lceil \lambda s.\ s{\in}S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
by (*rule R-g-ode-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-ode-rule*: $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode*[*symmetric*] **by** (*rule rel-rkat.R2*)

lemma *R-g-odel*: $P = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0)$; *rel-R* $\lceil R\rceil\ \lceil Q\rceil \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
by (*rule R-g-odel-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-oder*: $R = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
rel-R $\lceil P\rceil\ \lceil R\rceil$; $(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
by (*rule R-g-oder-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-ode-ivl*:
$t \geq 0 \Longrightarrow t \in T \Longrightarrow (\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ \{0..t\})\ S\ @\ 0) \leq$ *rel-R* $\lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode-ivl*[*symmetric*] **by** (*rule rel-rkat.R2*)

end

— Evolution command (invariants)

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

0.9.5 Derivation of the rules of dL

We derive a generalised version of some domain specific rules of differential dynamic logic (dL).

abbreviation *g-dl-ode* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a rel* ((*1x*´=- & -))
where $(x'{=}f\ \&\ G) \equiv (x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ \{t.\ t \geq 0\})\ \textit{UNIV}\ @\ 0)$

abbreviation *g-dl-ode-inv* :: (('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a pred* ⇒ '*a rel* ((*1x*´=- & - *DINV* -))
where $(x' = f\ \&\ G\ \textit{DINV}\ I) \equiv (x' = (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ \{t.\ t \geq 0\})\ \textit{UNIV}\ @\ 0\ \textit{DINV}\ I)$

lemma *diff-solve-rule1*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows *rel-kat.Hoare* $\lceil P\rceil\ (x' = f\ \&\ G)\ \lceil Q\rceil$
using *assms* **by**(*subst local-flow.sH-g-ode-subset*, *auto*)

lemma *diff-solve-rule2*:
fixes *c*::'*a*::{*heine-borel*, *banach*}
assumes $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow Q\ (s + t *_R c))$
shows *rel-kat.Hoare* $\lceil P\rceil\ (x'{=}(\lambda s.\ c)\ \&\ G)\ \lceil Q\rceil$
apply(*subst local-flow.sH-g-ode-subset*[**where** $T{=}\textit{UNIV}$ **and** $\varphi{=}(\lambda\ t\ x.\ x + t *_R c)$])
using *line-is-local-flow assms* **by** *auto*

lemma *diff-weak-rule*:
  assumes $\lceil G\rceil \leq \lceil Q\rceil$
  shows *rel-kat.Hoare* $\lceil P\rceil$ $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0)$ $\lceil Q\rceil$
  using *assms* **unfolding** *g-orbital-eq rel-kat-H ivp-sols-def g-ode-def* **by** *auto*

lemma *diff-cut-rule*:
  assumes *wp-C*:*rel-kat.Hoare* $\lceil P\rceil$ $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0)$ $\lceil C\rceil$
    and *wp-Q*:*rel-kat.Hoare* $\lceil P\rceil$ $(x' = f \ \& \ (\lambda \ s. \ G \ s \wedge C \ s) \ on \ U \ S \ @ \ t_0)$ $\lceil Q\rceil$
  shows *rel-kat.Hoare* $\lceil P\rceil$ $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0)$ $\lceil Q\rceil$
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** $t \in U \ s$
    and *x-ivp*:$X \in$ *ivp-sols* $f \ U \ S \ t_0 \ s$
    and *guard-x*:$\forall x. \ x \in U \ s \wedge x \leq t \longrightarrow G \ (X \ x)$
  have $\forall \ t \in (down \ (U \ s) \ t). \ X \ t \in$ *g-orbital* $f \ G \ U \ S \ t_0 \ s$
    using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
  hence $\forall \ t \in (down \ (U \ s) \ t). \ C \ (X \ t)$
    using *wp-C* ‹$P \ s$› **by** (*subst* (*asm*) *rel-kat-H*, *auto simp*: *g-ode-def*)
  hence $X \ t \in$ *g-orbital* $f \ (\lambda s. \ G \ s \wedge C \ s) \ U \ S \ t_0 \ s$
    using *guard-x* ‹$t \in U \ s$› **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

lemma *diff-inv-rule*:
  assumes $\lceil P\rceil \leq \lceil I\rceil$ **and** *diff-invariant* $I \ f \ U \ S \ t_0 \ G$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
  shows *rel-kat.Hoare* $\lceil P\rceil$ $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0)$ $\lceil Q\rceil$
  apply(*subst g-ode-inv-def*[*symmetric*, **where** $I = I$], *rule H-g-ode-inv*)
  unfolding *sH-diff-inv* **using** *assms* **by** *auto*

end

0.9.6 Examples

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

theory *HS-VC-KAT-Examples-rel*
  imports *HS-VC-KAT-rel*

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*ˆ*2* $\Rightarrow$ *real*ˆ*2* (f)
  where $f \ s \equiv (\chi \ i. \ if \ i{=}1 \ then \ s\$2 \ else \ -s\$1)$

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

— Verified with annotated dynamics

lemma *pendulum-dyn*: *rel-kat.Hoare* $\lceil \lambda s. \ r^2 = (s\$1)^2 + (s\$2)^2\rceil$ $(EVOL \ \varphi \ G \ T)$ $\lceil \lambda s. \ r^2 = (s\$1)^2 + (s\$2)^2\rceil$

by *simp*

— Verified with differential invariants

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

— Verified with the flow

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

lemma *pendulum-flow*: *rel-kat.Hoare*
$\lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil\ (x\acute{}{=}f\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*subst local-flow.sH-g-ode-subset*[*OF local-flow-pend*], *simp-all*)

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

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

— 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 $(x{::}real) \leq h$
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 ≡ (λs. 2 · g · s$1 − 2 · g · h − (s$2 · s$2) = 0)
  shows diff-invariant I (λt. f g) (λs. UNIV) UNIV 0 G
  unfolding dinv apply(rule diff-invariant-rules, simp)
  by(auto intro!: poly-derivatives)

lemma bouncing-ball-inv: g < 0 ⟹ h ≥ 0 ⟹ rel-kat.Hoare
  ⌈λs. s$1 = h ∧ s$2 = 0⌉
  (LOOP
      ((x´= f g & (λ s. s$1 ≥ 0) DINV (λs. 2 · g · s$1 − 2 · g · h − s$2 · s$2 = 0));
       (IF (λ s. s$1 = 0) THEN (2 ::= (λs. − s$2)) ELSE skip))
    INV (λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
  ) ⌈λs. 0 ≤ s$1 ∧ s$1 ≤ h⌉
  apply(rule H-loopI)
    apply(rule H-seq[where R=λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · 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 · g · x = 2 · g · h + v · v
    and pos: g · τ^2 / 2 + v · τ + (x::real) = 0
  shows 2 · g · h + (− (g · τ) − v) · (− (g · τ) − v) = 0
    and 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 · 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
    by(auto simp: algebra-simps 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 ⟹ rel-kat.Hoare
  ⌈λs. s$1 = h ∧ s$2 = 0⌉
  (LOOP
     ((EVOL (φ g) (λ s. s$1 ≥ 0) T);
      (IF (λ s. s$1 = 0) THEN (2 ::= (λs. − s$2)) ELSE skip))
   INV (λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
  ) ⌈λs. 0 ≤ s$1 ∧ s$1 ≤ h⌉
  apply(rule H-loopI, rule H-seq[where R=λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2])
  by (auto simp: bb-real-arith)
```

— Verified with the flow

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

lemma bouncing-ball-flow: g < 0 ⟹ h ≥ 0 ⟹ rel-kat.Hoare
  ⌈λs. s$1 = h ∧ s$2 = 0⌉
  (LOOP
     ((x´= f g & (λ s. s$1 ≥ 0));
      (IF (λ s. s$1 = 0) THEN (2 ::= (λs. − s$2)) ELSE skip))
   INV (λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
  ) ⌈λs. 0 ≤ s$1 ∧ s$1 ≤ h⌉
  apply(rule H-loopI)
   apply(rule H-seq[where R=λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2])
    apply(subst local-flow.sH-g-ode-subset[OF local-flow-ball], simp)
    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) ⟹ 0 ≤ h ⟹
  2 ::= (λs. − s$2) ≤ rel-R
   ⌈λs. s$1 = 0 ∧ 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2⌉
   ⌈λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2⌉
  by (rule R-assign-rule, auto)

lemma R-bouncing-ball-dyn:
  assumes g < 0 and h ≥ 0
  shows rel-R ⌈λs. s$1 = h ∧ s$2 = 0⌉ ⌈λs. 0 ≤ s$1 ∧ s$1 ≤ h⌉ ≥
  (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))
  apply(rule order-trans)
   apply(rule R-loop-mono) defer
   apply(rule R-loop)
    apply(rule R-seq)
  using assms apply(simp-all, force simp: bb-real-arith)
  apply(rule R-seq-mono) defer
  apply(rule order-trans)
   apply(rule R-cond-mono) defer defer
    apply(rule R-cond) defer
  using R-bb-assign apply force
   apply(rule R-skip, clarsimp)
  by (rule R-g-evol-rule, force simp: bb-real-arith)
```

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

Thermostat

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

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

abbreviation *therm-guard* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (G)
  where $G\ Tmin\ Tmax\ a\ L\ s \equiv (s\$2 \leq - (ln\ ((L-(if\ L{=}0\ then\ Tmin\ else\ Tmax))/(L-s\$3)))/a)$

abbreviation *therm-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (I)
  where $I\ Tmin\ Tmax\ s \equiv Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *therm-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *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*: $0 < Tmin\ Tmin \leq T\ T \leq Tmax$
    and *thyps*: $0 \leq (\tau{::}real)\ \forall\tau\in\{0..\tau\}.\ \tau \leq - (ln\ (Tmin\ /\ T)\ /\ a)$
  shows $Tmin \leq exp\ (-a * \tau) * T$ **and** $exp\ (-a * \tau) * T \leq Tmax$
proof−

have *0* $\leq \tau \wedge \tau \leq -$ (*ln* (*Tmin* / *T*) / *a*)
  using *thyps* **by** *auto*
hence *ln* (*Tmin* / *T*) $\leq -$ *a* $* \tau \wedge -$ *a* $* \tau \leq$ *0*
  using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have *Tmin* / *T* > *0*
  using *Thyps* **by** *auto*
ultimately have *obs*: *Tmin* / *T* $\leq$ *exp* ($-a * \tau$) *exp* ($-a * \tau$) $\leq$ *1*
  using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*, *simp*)
thus *Tmin* $\leq$ *exp* ($-a * \tau$) $*$ *T*
  using *Thyps* **by** (*simp add*: *pos-divide-le-eq*)
show *exp* ($-a * \tau$) $*$ *T* $\leq$ *Tmax*
  using *Thyps mult-left-le-one-le*[*OF - exp-ge-zero obs*(*2*), *of T*]
    less-eq-real-def order-trans-rules(*23*) **by** *blast*
qed

lemma *therm-dyn-up-real-arith*:
  assumes *a* > *0* **and** *Thyps*: *Tmin* $\leq$ *T T* $\leq$ *Tmax Tmax* < (*L*::*real*)
    and *thyps*: *0* $\leq \tau\ \forall\tau\in\{0..\tau\}.\ \tau \leq -$ (*ln* ((*L* $-$ *Tmax*) / (*L* $-$ *T*)) / *a*)
  shows *L* $-$ *Tmax* $\leq$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*)
    and *L* $-$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*) $\leq$ *Tmax*
    and *Tmin* $\leq$ *L* $-$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*)
proof−
  have *0* $\leq \tau \wedge \tau \leq -$ (*ln* ((*L* $-$ *Tmax*) / (*L* $-$ *T*)) / *a*)
    using *thyps* **by** *auto*
  hence *ln* ((*L* $-$ *Tmax*) / (*L* $-$ *T*)) $\leq -$ *a* $* \tau \wedge -$ *a* $* \tau \leq$ *0*
    using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
  also have (*L* $-$ *Tmax*) / (*L* $-$ *T*) > *0*
    using *Thyps* **by** *auto*
  ultimately have (*L* $-$ *Tmax*) / (*L* $-$ *T*) $\leq$ *exp* ($-a * \tau$) $\wedge$ *exp* ($-a * \tau$) $\leq$ *1*
    using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
  moreover have *L* $-$ *T* > *0*
    using *Thyps* **by** *auto*
  ultimately have *obs*: (*L* $-$ *Tmax*) $\leq$ *exp* ($-a * \tau$) $*$ (*L* $-$ *T*) $\wedge$ *exp* ($-a * \tau$) $*$ (*L* $-$ *T*) $\leq$ (*L* $-$ *T*)
    by (*simp add*: *pos-divide-le-eq*)
  thus (*L* $-$ *Tmax*) $\leq$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*)
    by *auto*
  thus *L* $-$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*) $\leq$ *Tmax*
    by *auto*
  show *Tmin* $\leq$ *L* $-$ *exp* ($-(a * \tau)$) $*$ (*L* $-$ *T*)
    using *Thyps* **and** *obs* **by** *auto*
qed

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

lemma *thermostat-flow*:
  assumes *0* < *a* **and** *0* $\leq \tau$ **and** *0* < *Tmin* **and** *Tmax* < *L*
  shows *rel-kat.Hoare* ⌈*I Tmin Tmax*⌉
  (*LOOP* (
    — control
    (*2* ::= ($\lambda s.$ *0*));
    (*3* ::= ($\lambda s.$ *s*\$*1*));
    (*IF* ($\lambda s.$ *s*\$*4* = *0* $\wedge$ *s*\$*3* $\leq$ *Tmin* + *1*) *THEN*
      (*4* ::= ($\lambda s.$*1*))
     ELSE IF ($\lambda s.$ *s*\$*4* = *1* $\wedge$ *s*\$*3* $\geq$ *Tmax* $-$ *1*) *THEN*
      (*4* ::= ($\lambda s.$*0*))
     ELSE skip);
    — dynamics
    (*IF* ($\lambda s.$ *s*\$*4* = *0*) *THEN*
      (*x*´= ($\lambda t.$ *f a 0*) & *G Tmin Tmax a 0 on* ($\lambda s.$ {*0*..τ}) *UNIV* @ *0*)
    ELSE

```
    (x´= (λt. f a L) & G Tmin Tmax a L on (λs. {0..τ}) UNIV @ 0))
  ) INV I Tmin Tmax)
  ⌈I Tmin Tmax⌉
  apply(rule H-loopI)
    apply(rule-tac R=λs. I Tmin Tmax s ∧ s$2=0 ∧ s$3 = s$1 in H-seq)
     apply(rule-tac R=λs. I Tmin Tmax s∧ s$2=0 ∧ s$3 = s$1 in H-seq)
      apply(rule-tac R=λs. 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 ⌈λs. s$4 = 0 ∧ I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉ ≥
    (x´= (λt. f a 0) & G Tmin Tmax a 0 on (λs. {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 ⌈λs. s$4 ≠ 0 ∧ I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉ ≥
    (x´= (λt. f a L) & G Tmin Tmax a L on (λs. {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 ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉ ≥
  (IF (λs. s$4 = 0) THEN
    (x´= (λt. f a 0) & G Tmin Tmax a 0 on (λs. {0..τ}) UNIV @ 0)
  ELSE
    (x´= (λt. f a L) & G Tmin Tmax a L on (λs. {0..τ}) UNIV @ 0))
  apply(rule order-trans, rule R-cond-mono)
  apply(rule R-therm-dyn-down[OF assms])
  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⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0⌉ ≥ (2 ::= (λs. 0))
  by (auto simp: R-assign-rule)

lemma R-therm-assign2:
  rel-R ⌈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:
  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* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil I\ Tmin\ Tmax \rceil \geq$
($LOOP$
rel-R $\lceil I\ Tmin\ Tmax \rceil$ $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$;
rel-R $\lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil$ $\lceil I\ Tmin\ Tmax \rceil$
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 *rel-R* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil I\ Tmin\ Tmax \rceil \geq$
(*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*
$(x' = (\lambda t.\ f\ a\ 0)\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0)$
ELSE
$(x' = (\lambda t.\ f\ a\ L)\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0))$
) *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* :: $real \Rightarrow real\hat{\ }4 \Rightarrow real\hat{\ }4$ (f)
where $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

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

abbreviation *tank-guard* :: $real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (G)
where $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: $real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (I)
where $I\ hmin\ hmax\ s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *tank-diff-inv* :: $real \Rightarrow real \Rightarrow real \Rightarrow real\hat{\ }4 \Rightarrow bool$ (dI)
where $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$
$hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

— Verified with the flow

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

lemma *tank-arith*:
  assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** $c_o < c_i$
  shows $\forall\tau\in\{0..\tau\}.\ \tau \leq -((hmin - y)\ /\ c_o) \Longrightarrow hmin \leq y - c_o * \tau$
    and $\forall\tau\in\{0..\tau\}.\ \tau \leq (hmax - y)\ /\ (c_i - c_o) \Longrightarrow (c_i - c_o) * \tau + y \leq hmax$
    and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) \cdot \tau + y$
    and $y \leq hmax \Longrightarrow y - c_o \cdot \tau \leq hmax$
  apply(*simp-all add*: *field-simps le-divide-eq assms*)
  using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
  using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

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

lemma *tank-flow*:
  assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
  shows *rel-kat.Hoare* ⌈*I hmin hmax*⌉
  (*LOOP*
    — control
    ((*2* ::=(λ*s.0*));(*3* ::=(λ*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' = (\lambda t.\ f\ (c_i - c_o))$ & *G hmax* $(c_i - c_o)$ *on* ($\lambda s.\ \{0..\tau\}$) *UNIV* @ *0*)
     ELSE ($x' = (\lambda t.\ f\ (-c_o))$ & *G hmin* $(-c_o)$ *on* ($\lambda s.\ \{0..\tau\}$) *UNIV* @ *0*)))
  INV I hmin hmax) ⌈*I hmin hmax*⌉
  apply(*rule H-loopI*)
    apply(*rule-tac R=*$\lambda s.$ *I hmin hmax s* $\wedge\ s\$2=0 \wedge s\$3 = s\$1$ **in** *H-seq*)
     apply(*rule-tac R=*$\lambda s.$ *I hmin hmax s* $\wedge\ s\$2=0 \wedge s\$3 = s\$1$ **in** *H-seq*)
      apply(*rule-tac R=*$\lambda s.$ *I hmin hmax s* $\wedge\ s\$2=0$ **in** *H-seq*, *simp*, *simp*)
     apply(*rule H-cond*, *simp-all add*: *H-g-ode-tank*[*OF assms*(*1*)])
  using *assms tank-arith*[*OF - assms*(*2*,*3*)] **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:
  $0 \leq \tau \Longrightarrow$ *diff-invariant* (*dI hmin hmax k*) ($\lambda t.\ f\ k$) ($\lambda s.\ \{0..\tau\}$) *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*, *presburger*)
    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 mult-nonneg-nonneg*)
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*´= (λ*t*. *f* ($c_i - c_o$)) & *G hmax* ($c_i - c_o$) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$)))
ELSE
(*x*´= (λ*t*. *f* ($-c_o$)) & *G hmin* ($-c_o$) *on* (λ*s*. {*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*´= (λ*t*. *f* ($c_i - c_o$)) & *G hmax* ($c_i - c_o$) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$)))
ELSE
(*x*´= (λ*t*. *f* ($-c_o$)) & *G hmin* ($-c_o$) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* ($-c_o$)))))
INV I hmin hmax) (**is** *LOOP* (*?ctrl*;*?dyn*) *INV* - ≤ *?ref*)
proof−
— First we refine the control.
let *?Icntrl* = λ*s*. *I hmin hmax s* ∧ *s*\$*2* = *0* ∧ *s*\$*3* = *s*\$*1*

and *?cond* = $\lambda s.$ *s$4 = 0* ∧ *s$3* ≤ *hmin + 1*
have *ifbranch1*: *4* ::= ($\lambda s.1$) ≤ *rel-R* ⌈$\lambda s.$ *?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF* ($\lambda s.$ *s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= ($\lambda s.0$)) *ELSE skip*) ≤
rel-R ⌈$\lambda s.$ ¬ *?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch2*)
apply(*rule order-trans*, *rule R-cond-mono*) **defer defer**
by (*rule R-cond*) (*auto intro*!: *R-assign-rule R-skip*)
have *ifthenelse*: (*IF ?cond THEN ?branch1 ELSE ?branch2*) ≤ *rel-R* ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉ (**is** *?ifthenelse* ≤ -)
by (*rule R-cond*)
have (*IF ?cond THEN* (*4* ::= ($\lambda s.1$)) *ELSE* (*IF* ($\lambda s.$ *s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= ($\lambda s.0$)) *ELSE skip*)) ≤
rel-R ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉
apply(*rule-tac y=?ifthenelse* **in** *order-trans*, *rule R-cond-mono*)
using *ifbranch1 ifbranch2 ifthenelse* **by** *auto*
hence *ctrl*: *?ctrl* ≤ *rel-R* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉
apply(*rule-tac R=?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* ∧ *s$2 = 0* **in** *R-seq-rule*)
by (*auto intro*!: *R-assign-rule*)
— Then we refine the dynamics.
have *dynup*: (*x´*= ($\lambda t.$ *f* ($c_i - c_o$)) & *G hmax* ($c_i - c_o$) *on* ($\lambda s.$ {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* ($c_i - c_o$))) ≤
rel-R ⌈$\lambda s.$ *s$4 = 0* ∧ *?Icntrl s*⌉ ⌈*I hmin hmax*⌉
apply(*rule R-g-ode-inv*[*OF tank-diff-inv*[*OF assms*(*1*)]])
using *assms* **by** (*auto simp*: *tank-inv-arith1*)
have *dyndown*: (*x´*= ($\lambda t.$ *f* ($-c_o$)) & *G hmin* ($-c_o$) *on* ($\lambda s.$ {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* ($-c_o$))) ≤
rel-R ⌈$\lambda s.$ *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*⌉ ≤ ⌈$\lambda s.$ *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

0.10 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 *HS-VC-KAT-ndfun*
  imports
    HS-VC-KAT
    ../HS-ODEs
    Transformer-Semantics.Kleisli-Quantale

begin

0.10.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

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

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

no-notation *Archimedean-Field.ceiling* ($\lceil - \rceil$)
  and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor - \rfloor$)
  and *tau* (τ)
  and *Relation.relcomp* (**infixl** ; 75)
  and *proto-near-quantale-class.bres* (**infixr** $\rightarrow$ 60)

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

thm *sup-nd-fun-def sup-fun-def*

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

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

lemma *nd-fun-distr*[*nd-fun-aka*]: $(x + y) \cdot z = x \cdot z + y \cdot z$

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

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

lemma *nd-fun-leq*[*nd-fun-aka*]: $(x \leq y) = (x + y = y)$
and *nd-fun-less*[*nd-fun-aka*]: $(x < y) = (x + y = y \wedge x \neq y)$
and *nd-fun-leq-add*[*nd-fun-aka*]: $z \cdot x \leq z \cdot (x + y)$ **for** x::$'a$ *nd-fun*
unfolding *less-eq-nd-fun-def less-nd-fun-def plus-nd-fun-def times-nd-fun-def sup-fun-def*
by (*unfold nd-fun-eq-iff le-fun-def*, *auto simp*: *kcomp-def*)

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

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

end

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

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

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

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

end

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

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

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

end

— 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*]:
$\lceil P \rceil \leq \lceil Q \rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
$(\lceil P \rceil = \lceil Q \rceil) = (\forall s.\ P\ s = Q\ s)$
$(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s.\ P\ s \wedge Q\ s \rceil$
$(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s.\ P\ s \vee Q\ s \rceil$
$tt\ \lceil P \rceil = \lceil P \rceil$
$n\ \lceil P \rceil = \lceil \lambda s.\ \neg\ P\ s \rceil$
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 $skip \equiv (1::'a\ nd\text{-}fun)$

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

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

lemma *H-assign*: $P = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil$
unfolding *ndfun-kat-H assign-def vec-upd-def* **by** *force*

lemma *sH-assign*[*simp*]: $Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil \longleftrightarrow (\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j))$
unfolding *ndfun-kat-H vec-upd-def assign-def* **by** (*auto simp*: *fun-upd-def*)

— Next, the Hoare rule of the composition

abbreviation $seq\text{-}seq :: 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun$ (**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 $cond\text{-}sugar :: 'a\ pred \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun$ (*IF - THEN - ELSE -* [*64*,*64*] *63*)
where $IF\ B\ THEN\ X\ ELSE\ Y \equiv kat\text{-}cond\ \lceil B \rceil\ X\ Y$

lemma *H-cond*: $Hoare\ \lceil \lambda s.\ P\ s \wedge B\ s \rceil\ X\ \lceil Q \rceil \Longrightarrow Hoare\ \lceil \lambda s.\ P\ s \wedge \neg\ B\ s \rceil\ Y\ \lceil Q \rceil \Longrightarrow$
$Hoare\ \lceil P \rceil\ (IF\ B\ THEN\ X\ ELSE\ Y)\ \lceil Q \rceil$

by (*rule H-cond*, *simp-all*)

lemma *sH-cond*[*simp*]: *Hoare* $\lceil P \rceil$ (*IF B THEN X ELSE Y*) $\lceil Q \rceil \longleftrightarrow$
(*Hoare* $\lceil \lambda s.\ P\ s \wedge B\ s \rceil$ *X* $\lceil Q \rceil$ $\wedge$ *Hoare* $\lceil \lambda s.\ P\ s \wedge \neg\ B\ s \rceil$ *Y* $\lceil Q \rceil$)
by (*auto simp*: *H-cond-iff ndfun-kat-H*)

— Rewriting the Hoare rule for the while loop

abbreviation *while-inv-sugar* :: *'a pred* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a nd-fun* $\Rightarrow$ *'a nd-fun* (*WHILE - INV - DO -* [*64*,*64*,*64*] *63*)
where *WHILE B INV I DO X* $\equiv$ *kat-while-inv* $\lceil B \rceil$ $\lceil I \rceil$ *X*

lemma *sH-while-inv*: $\forall s.\ P\ s \longrightarrow I\ s \Longrightarrow \forall s.\ I\ s \wedge \neg\ B\ s \longrightarrow Q\ s \Longrightarrow$ *Hoare* $\lceil \lambda s.\ I\ s \wedge B\ s \rceil$ *X* $\lceil I \rceil$
$\Longrightarrow$ *Hoare* $\lceil P \rceil$ (*WHILE B INV I DO X*) $\lceil Q \rceil$
by (*rule H-while-inv*, *simp-all add*: *ndfun-kat-H*)

— Finally, we add a Hoare triple rule for finite iterations.

abbreviation *loopi-sugar* :: *'a nd-fun* $\Rightarrow$ *'a pred* $\Rightarrow$ *'a nd-fun* (*LOOP - INV -* [*64*,*64*] *63*)
where *LOOP X INV I* $\equiv$ *kat-loop-inv X* $\lceil I \rceil$

lemma *H-loop*: *Hoare* $\lceil P \rceil$ *X* $\lceil P \rceil \Longrightarrow$ *Hoare* $\lceil P \rceil$ (*LOOP X INV I*) $\lceil P \rceil$
by (*auto intro*: *H-loop*)

lemma *H-loopI*: *Hoare* $\lceil I \rceil$ *X* $\lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow$ *Hoare* $\lceil P \rceil$ (*LOOP X INV I*) $\lceil Q \rceil$
using *H-loop-inv*[*of* $\lceil P \rceil$ $\lceil I \rceil$ *X* $\lceil Q \rceil$] **by** *auto*

0.10.2 Verification of hybrid programs

— Verification by providing evolution

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

lemma *sH-g-evol*[*simp*]:
fixes φ :: (*'a::preorder*) $\Rightarrow$ *'b* $\Rightarrow$ *'b*
shows *Hoare* $\lceil P \rceil$ (*EVOL* φ *G U*) $\lceil Q \rceil$ = ($\forall s.\ P\ s \longrightarrow$ ($\forall t{\in}U\ s.$ ($\forall \tau{\in}$*down* (*U s*) *t*. *G* ($\varphi\ \tau\ s$)) $\longrightarrow$ *Q* ($\varphi\ t\ s$)))
unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** *auto*

lemma *H-g-evol*:
fixes φ :: (*'a::preorder*) $\Rightarrow$ *'b* $\Rightarrow$ *'b*
assumes *P* = ($\lambda s.$ ($\forall t{\in}U\ s.$ ($\forall \tau{\in}$*down* (*U s*) *t*. *G* ($\varphi\ \tau\ s$)) $\longrightarrow$ *Q* ($\varphi\ t\ s$)))
shows *Hoare* $\lceil P \rceil$ (*EVOL* φ *G U*) $\lceil Q \rceil$
by (*simp add*: *assms*)

— Verification by providing solutions

definition *g-ode* ::(*real* $\Rightarrow$ (*'a::banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a pred* $\Rightarrow$ (*'a* $\Rightarrow$ *real set*) $\Rightarrow$ *'a set* $\Rightarrow$
real $\Rightarrow$ *'a nd-fun* ((*1x´= - & - on - - @ -*))
where (*x´= f & G on U S @* t_0) $\equiv$ (λ *s*. *g-orbital f G U S* t_0 *s*)$^\bullet$

lemma *H-g-orbital*:
P = ($\lambda s.$ ($\forall X{\in}$*ivp-sols f U S* t_0 *s*. $\forall t{\in}U\ s.$ ($\forall \tau{\in}$*down* (*U s*) *t*. *G* (*X* τ)) $\longrightarrow$ *Q* (*X t*))) $\Longrightarrow$
Hoare $\lceil P \rceil$ (*x´= f & G on U 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 U S @* t_0) $\lceil Q \rceil$ =
($\forall s.\ P\ s \longrightarrow$ ($\forall X{\in}$*ivp-sols f U S* t_0 *s*. $\forall t{\in}U\ s.$ ($\forall \tau{\in}$*down* (*U s*) *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 *sH-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
shows $\textit{Hoare}\ \lceil P\rceil\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ \textit{on}\ U\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}\textit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *clarsimp*, *safe*)
fix *s t*
assume *hyps*: $s \in S\ P\ s\ t \in U\ s\ \forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)$
and *main*: $\forall s.\ P\ s \longrightarrow (\forall X{\in}\textit{Sols}\ (\lambda t.\ f)\ U\ S\ 0\ s.\ \forall t{\in}U\ s.\ (\forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
hence $(\lambda t.\ \varphi\ t\ s) \in \textit{Sols}\ (\lambda t.\ f)\ U\ S\ 0\ s$
using *in-ivp-sols assms* **by** *blast*
thus $Q\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*
next
fix *s X t*
assume *hyps*: $P\ s\ X \in \textit{Sols}\ (\lambda t.\ f)\ U\ S\ 0\ s\ t \in U\ s\ \ \forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (X\ \tau)$
and *main*: $\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau.\ \tau \in U\ s \wedge \tau \le t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
hence *obs*: $s \in S$
using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
hence $\forall \tau{\in}\textit{down}\ (U\ s)\ t.\ X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps assms* **by** *blast*
thus $Q\ (X\ t)$
using *hyps main obs* **by** *auto*
qed

lemma *H-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ s \in S \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}\textit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows $\textit{Hoare}\ \lceil P\rceil\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ \textit{on}\ U\ S\ @\ 0)\ \lceil Q\rceil$
using *assms* **apply**(*subst sH-g-ode-subset*[*OF assms*(*1*)])
unfolding *assms* **by** *auto*

lemma *sH-g-ode*: $\textit{Hoare}\ \lceil P\rceil\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ T)\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}\textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
by (*subst sH-g-ode-subset*, *auto simp*: *init-time interval-time*)

lemma *sH-g-ode-ivl*: $t \ge 0 \Longrightarrow t \in T \Longrightarrow \textit{Hoare}\ \lceil P\rceil\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ \textit{on}\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
apply(*subst sH-g-ode-subset*; *clarsimp*, (*force*)?)
using *init-time interval-time mem-is-interval-1-I* **by** *blast*

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

end

— Verification with differential invariants

definition *g-ode-inv* :: $(\textit{real} \Rightarrow ('a{::}\textit{banach}){\Rightarrow}'a) \Rightarrow 'a\ \textit{pred} \Rightarrow ('a \Rightarrow \textit{real set}) \Rightarrow 'a\ \textit{set} \Rightarrow$
$\textit{real} \Rightarrow 'a\ \textit{pred} \Rightarrow 'a\ \textit{nd-fun}$ ((*1x´=- & - on - - @ - DINV -*))
where $(x\acute{}= f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0\ \textit{DINV}\ I) = (x\acute{}= f\ \&\ G\ \textit{on}\ U\ S\ @\ t_0)$

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

lemma *sH-g-orbital-inv*:
assumes $\lceil P\rceil \le \lceil I\rceil$ **and** *Hoare* $\lceil I\rceil$ $(x\acute{} = f$ & G *on* T S @ $t_0)$ $\lceil I\rceil$ **and** $\lceil I\rceil \le \lceil Q\rceil$
shows *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** *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* $\lceil I\rceil$ $(x\acute{} = f$ & 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* $\lceil I\rceil$ $(x\acute{} = f$ & G *on* T S @ $t_0)$ $\lceil I\rceil \Longrightarrow \lceil P\rceil \le \lceil I\rceil \Longrightarrow$
$\lceil\lambda s.\ I\ s \wedge G\ s\rceil \le \lceil Q\rceil \Longrightarrow$ *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** *H-consr*, *simp*)
apply(*subst sH-g-orbital-guard*[*symmetric*], *force*)
by (*rule-tac* $I=I$ **in** *sH-g-orbital-inv*, *simp-all*)

0.10.3 Refinement Components

— Skip

lemma *R-skip*: $(\forall s.\ P\ s \longrightarrow Q\ s) \Longrightarrow 1 \le$ *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* $\lceil R\rceil$ $\lceil Q\rceil$) $\le$ *Ref* $\lceil P\rceil$ $\lceil Q\rceil$
using *R-seq* **by** *blast*

lemma *R-seq-rule*: $X \le$ *Ref* $\lceil P\rceil$ $\lceil R\rceil \Longrightarrow Y \le$ *Ref* $\lceil R\rceil$ $\lceil Q\rceil \Longrightarrow X;\ Y \le$ *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*: $(x ::= e) \le$ *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*: $(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \le$ *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* $\lceil R\rceil$ $\lceil Q\rceil \le$ *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* $\lceil P\rceil$ $\lceil R\rceil$; $(x ::= e) \le$ *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* $\lceil Q\rceil$ $\lceil Q\rceil \le$ *Ref* $\lceil(\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j))\rceil$ $\lceil Q\rceil$
by (*rule R-assignl*) *simp*

lemma *Ref* $\lceil Q\rceil$ $\lceil(\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j))\rceil$; $(x ::= e) \le$ *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* $\lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil$ $\lceil Q\rceil$) $\le$ *Ref* $\lceil P\rceil$ $\lceil Q\rceil$
using *R-cond*[*of* $\lceil B\rceil$ $\lceil P\rceil$ $\lceil Q\rceil$] **by** *simp*

lemma *R-cond-mono*: $X \leq X' \Longrightarrow Y \leq Y' \Longrightarrow (IF\ P\ THEN\ X\ ELSE\ Y) \leq IF\ P\ THEN\ X'\ ELSE\ Y'$
unfolding *kat-cond-def times-nd-fun-def plus-nd-fun-def n-op-nd-fun-def*
by (*auto simp*: *kcomp-def less-eq-nd-fun-def p2ndf-def le-fun-def*)

— While loop

lemma *R-while*: $WHILE\ Q\ INV\ I\ DO\ (Ref\ \lceil\lambda s.\ P\ s \wedge Q\ s\rceil\ \lceil P\rceil) \leq Ref\ \lceil P\rceil\ \lceil\lambda s.\ P\ s \wedge \neg\ Q\ s\rceil$
unfolding *kat-while-inv-def* **using** *R-while*[*of* $\lceil Q\rceil$ $\lceil P\rceil$] **by** *simp*

lemma *R-while-mono*: $X \leq X' \Longrightarrow (WHILE\ P\ INV\ I\ DO\ X) \leq WHILE\ P\ INV\ I\ DO\ X'$
by (*simp add*: *kat-while-inv-def kat-while-def mult-isol mult-isor star-iso*)

— Finite loop

lemma *R-loop*: $X \leq Ref\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow LOOP\ X\ INV\ I \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \leq X' \Longrightarrow LOOP\ X\ INV\ I \leq LOOP\ X'\ INV\ I$
unfolding *kat-loop-inv-def* **by** (*simp add*: *star-iso*)

— Evolution command (flow)

lemma *R-g-evol*:
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
shows $(EVOL\ \varphi\ G\ U) \leq Ref\ \lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)\rceil\ \lceil P\rceil$
unfolding *spec-def* **by** (*rule H-g-evol*, *simp*)

lemma *R-g-evol-rule*:
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
shows $(\forall s.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (EVOL\ \varphi\ G\ U) \leq Ref$
$\lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-evol*[*symmetric*] *spec-def* **.**

lemma *R-g-evoll*:
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
shows $P = (\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(EVOL\ \varphi\ G\ U)\ ;\ Ref\ \lceil R\rceil\ \lceil Q\rceil \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
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 $R = (\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$Ref\ \lceil P\rceil\ \lceil R\rceil;\ (EVOL\ \varphi\ G\ U) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
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 $EVOL\ \varphi\ G\ U\ ;\ Ref\ \lceil Q\rceil\ \lceil Q\rceil \leq Ref\ \lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil$
$\lceil Q\rceil$
by (*rule R-g-evoll*) *simp*

lemma
fixes φ :: ($'a$::*preorder*) $\Rightarrow$ $'b$ $\Rightarrow$ $'b$
shows $Ref\ \lceil Q\rceil\ \lceil\lambda s.\ \forall t{\in}U\ s.\ (\forall\tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil;\ EVOL\ \varphi\ G\ U \leq Ref\ \lceil Q\rceil$
$\lceil Q\rceil$
by (*rule R-g-evolr*) *simp*

— Evolution command (ode)

context *local-flow*
begin

lemma *R-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
shows $(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq Ref\ \lceil\lambda s.\ s{\in}S \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
unfolding *spec-def* **by** (*rule H-g-ode-subset*[*OF assms*], *auto*)

lemma *R-g-ode-rule-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
shows $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *spec-def* **by** (*subst sH-g-ode-subset*[*OF assms*], *auto*)

lemma *R-g-odel-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s))$
shows $(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ ;\ Ref\ \lceil R\rceil\ \lceil Q\rceil \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply (*rule-tac R=R* **in** *R-seq-rule*, *rule-tac R-g-ode-rule-subset*)
by (*simp-all add*: *assms*)

lemma *R-g-oder-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge \textit{is-interval}\ (U\ s) \wedge U\ s \subseteq T$
and $R = (\lambda s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $Ref\ \lceil P\rceil\ \lceil R\rceil;\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
apply (*rule-tac R=R* **in** *R-seq-rule*, *simp*)
by (*rule-tac R-g-ode-rule-subset*, *simp-all add*: *assms*)

lemma *R-g-ode*: $(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq Ref\ \lceil\lambda s.\ s{\in}S \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
by (*rule R-g-ode-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-ode-rule*: $(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode*[*symmetric*] **by** (*rule R2*)

lemma *R-g-odel*: $P = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ ;\ Ref\ \lceil R\rceil\ \lceil Q\rceil \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
by (*rule R-g-odel-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-oder*: $R = (\lambda s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$Ref\ \lceil P\rceil\ \lceil R\rceil;\ (x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
by (*rule R-g-oder-subset*, *auto simp*: *init-time interval-time*)

lemma *R-g-ode-ivl*:
$t \geq 0 \Longrightarrow t \in T \Longrightarrow (\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}\{0..t\}.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x\acute{}= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *sH-g-ode-ivl*[*symmetric*] **by** (*rule R2*)

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: $\textit{diff-invariant}\ I\ f\ T\ S\ t_0\ G \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil\lambda s.\ I\ s \wedge G\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$(x\acute{}=f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
unfolding *spec-def* **by** (*auto simp*: *H-g-ode-inv*)

0.10.4 Derivation of the rules of dL

We derive a generalised version of some domain specific rules of differential dynamic logic (dL).

abbreviation *g-dl-ode* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a nd-fun* ((1*x*´=- & -))
  where (*x*´=*f* & *G*) ≡ (*x*´= (λ*t*. *f*) & *G on* (λ*s*. {*t*. *t* ≥ *0*}) *UNIV* @ *0*)

abbreviation *g-dl-ode-inv* :: (('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a pred* ⇒ '*a nd-fun* ((1*x*´=- & - *DINV* -))
  where (*x*´= *f* & *G DINV I*) ≡ (*x*´= (λ*t*. *f*) & *G on* (λ*s*. {*t*. *t* ≥ *0*}) *UNIV* @ *0 DINV I*)

lemma *diff-solve-rule1*:
  assumes *local-flow f UNIV UNIV* φ
    and ∀ *s*. *P s* ⟶ (∀ *t*≥*0*. (∀ τ∈{*0*..*t*}. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))
  shows *Hoare* ⌈*P*⌉ (*x*´= *f* & *G*) ⌈*Q*⌉
  using *assms* **by**(*subst local-flow.sH-g-ode-subset*, *auto*)

lemma *diff-solve-rule2*:
  fixes *c*::'*a*::{*heine-borel*, *banach*}
  assumes ∀ *s*. *P s* ⟶ (∀ *t*≥*0*. (∀ τ∈{*0*..*t*}. *G* (*s* + τ $*_R$ *c*)) ⟶ *Q* (*s* + *t* $*_R$ *c*))
  shows *Hoare* ⌈*P*⌉ (*x*´=(λ*s*. *c*) & *G*) ⌈*Q*⌉
  apply(*subst local-flow.sH-g-ode-subset*[**where** *T*=*UNIV* **and** φ=(λ *t x*. *x* + *t* $*_R$ *c*)])
  using *line-is-local-flow assms* **by** *auto*

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

lemma *diff-cut-rule*:
  assumes *wp-C*:*Hoare* ⌈*P*⌉ (*x*´= *f* & *G on U S* @ t_0) ⌈*C*⌉
    and *wp-Q*:*Hoare* ⌈*P*⌉ (*x*´= *f* & (λ *s*. *G s* ∧ *C s*) *on U S* @ t_0) ⌈*Q*⌉
  shows *Hoare* ⌈*P*⌉ (*x*´= *f* & *G on U S* @ t_0) ⌈*Q*⌉
proof(*subst ndfun-kat-H*, *simp add*: *g-orbital-eq p2ndf-def g-ode-def*, *clarsimp*)
  fix *t*::*real* **and** *X*::*real* ⇒ '*a* **and** *s*
  assume *P s* **and** *t* ∈ *U s*
    and *x-ivp*:*X* ∈ *ivp-sols f U S* t_0 *s*
    and *guard-x*:∀ *x*. *x* ∈ *U s* ∧ *x* ≤ *t* ⟶ *G* (*X x*)
  have ∀ *t*∈(*down* (*U s*) *t*). *X t* ∈ *g-orbital f G U S* t_0 *s*
    using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
  hence ∀ *t*∈(*down* (*U s*) *t*). *C* (*X t*)
    using *wp-C* ‹*P s*› **by** (*subst* (*asm*) *ndfun-kat-H*, *auto simp*: *g-ode-def*)
  hence *X t* ∈ *g-orbital f* (λ*s*. *G s* ∧ *C s*) *U S* t_0 *s*
    using *guard-x* ‹*t* ∈ *U s*› **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

lemma *diff-inv-rule*:
  assumes ⌈*P*⌉ ≤ ⌈*I*⌉ **and** *diff-invariant I f U S* t_0 *G* **and** ⌈*I*⌉ ≤ ⌈*Q*⌉
  shows *Hoare* ⌈*P*⌉ (*x*´= *f* & *G on U S* @ t_0) ⌈*Q*⌉
  apply(*subst g-ode-inv-def*[*symmetric*, **where** *I*=*I*], *rule H-g-ode-inv*)
  unfolding *sH-diff-inv* **using** *assms* **by** *auto*

end

0.10.5 Examples

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

theory *HS-VC-KAT-Examples-ndfun*

imports *HS-VC-KAT-ndfun*

begin

Pendulum

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

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

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

— Verified with annotated dynamics

lemma *pendulum-dyn*: $Hoare\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil\ (EVOL\ \varphi\ G\ T)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by *simp*

— Verified with differential invariants

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

— Verified with the flow

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

lemma *pendulum-flow*: $Hoare\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil\ (x'{=}f\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2\rceil$
by (*subst local-flow.sH-g-ode-subset*[*OF local-flow-pend*], *simp-all*)

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

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

— 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 $(x::real) \leq h$
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 $(\lambda t.\ f\ g)$ $(\lambda s.\ UNIV)$ $UNIV\ 0\ G$
unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*)
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*
$((x\acute{} = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2 = 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ - s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
) $\lceil\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\rceil$
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* $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
    by(*auto simp*: *algebra-simps semiring-normalization-rules*(*29*))
  also have ... $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$ (**is** ... = *?middle*)
      by(*subst invar*, *simp*)
    finally have *?lhs* = *?middle*.
  moreover
  {**have** *?rhs* $= g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$
    by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
  also have ... = *?middle*
    by (*simp add*: *semiring-normalization-rules*(*29*))
  finally have *?rhs* = *?middle*.}
  ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *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*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$ *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-subset*[*OF local-flow-ball*], *simp*)
     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* $(\lambda t::real.\ f\ a\ L)$
  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*: $Tmin \le T\ T \le Tmax\ Tmax < (L::real)$
    and *thyps*: $0 \le \tau\ \forall\tau\in\{0..\tau\}.\ \tau \le -\ (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
  shows $L - Tmax \le exp\ (-(a * \tau)) * (L - T)$
    and $L - exp\ (-(a * \tau)) * (L - T) \le Tmax$
    and $Tmin \le L - exp\ (-(a * \tau)) * (L - T)$
proof−
  have $0 \le \tau \wedge \tau \le -\ (ln\ ((L - Tmax)\ /\ (L - T))\ /\ a)$
    using *thyps* **by** *auto*
  hence $ln\ ((L - Tmax)\ /\ (L - T)) \le -\ a * \tau \wedge -\ a * \tau \le 0$
    using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
  also have $(L - Tmax)\ /\ (L - T) > 0$
    using *Thyps* **by** *auto*
  ultimately have $(L - Tmax)\ /\ (L - T) \le exp\ (-a * \tau) \wedge exp\ (-a * \tau) \le 1$
    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*: $(L - Tmax) \le exp\ (-a * \tau) * (L - T) \wedge exp\ (-a * \tau) * (L - T) \le (L - T)$
    by (*simp add*: *pos-divide-le-eq*)
  thus $(L - Tmax) \le exp\ (-(a * \tau)) * (L - T)$
    by *auto*
  thus $L - exp\ (-(a * \tau)) * (L - T) \le Tmax$
    by *auto*
  show $Tmin \le L - exp\ (-(a * \tau)) * (L - T)$
    using *Thyps* **and** *obs* **by** *auto*
qed

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

lemma *thermostat-flow*:
assumes $0 < a$ **and** $0 \leq \tau$ **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* ∧ *s$3* ≤ *Tmin* + *1*) *THEN*
(*4* ::= ($\lambda s.$*1*))
ELSE IF ($\lambda s.$ *s$4* = *1* ∧ *s$3* ≥ *Tmax* − *1*) *THEN*
(*4* ::= ($\lambda s.$*0*))
ELSE skip);
— dynamics
(*IF* ($\lambda s.$ *s$4* = *0*) *THEN*
(*x´*= ($\lambda t.$ *f a 0*) & *G Tmin Tmax a 0 on* ($\lambda s.$ {*0..τ*}) *UNIV* @ *0*)
ELSE
(*x´*= ($\lambda t.$ *f a L*) & *G Tmin Tmax a L on* ($\lambda s.$ {*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* ∧ *s$2*=*0* ∧ *s$3* = *s$1* **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.$ *I Tmin Tmax s*∧ *s$2*=*0* ∧ *s$3* = *s$1* **in** *H-seq*)
apply(*rule-tac R*=$\lambda 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** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* ⌈$\lambda s.$ *s$4* = *0* ∧ *I Tmin Tmax s* ∧ *s$2* = *0* ∧ *s$3* = *s$1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x´*= ($\lambda t.$ *f a 0*) & *G Tmin Tmax a 0 on* ($\lambda s.$ {*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** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* ⌈$\lambda s.$ *s$4* ≠ *0* ∧ *I Tmin Tmax s* ∧ *s$2* = *0* ∧ *s$3* = *s$1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x´*= ($\lambda t.$ *f a L*) & *G Tmin Tmax a L on* ($\lambda s.$ {*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** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* ⌈$\lambda s.$ *I Tmin Tmax s* ∧ *s$2* = *0* ∧ *s$3* = *s$1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*IF* ($\lambda s.$ *s$4* = *0*) *THEN*
(*x´*= ($\lambda t.$ *f a 0*) & *G Tmin Tmax a 0 on* ($\lambda s.$ {*0..τ*}) *UNIV* @ *0*)
ELSE
(*x´*= ($\lambda t.$ *f a L*) & *G Tmin Tmax a L on* ($\lambda s.$ {*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*⌉ ⌈$\lambda s.$ *I Tmin Tmax s* ∧ *s$2* = *0*⌉ ≥ (*2* ::= ($\lambda s.$ *0*))
by (*auto simp*: *R-assign-rule*)

lemma *R-therm-assign2*:
Ref ⌈$\lambda s.$ *I Tmin Tmax s* ∧ *s$2* = *0*⌉ ⌈$\lambda s.$ *I Tmin Tmax s* ∧ *s$2* = *0* ∧ *s$3* = *s$1*⌉ ≥ (*3* ::= ($\lambda s.$ *s$1*))

by (*auto simp*: *R-assign-rule*)

lemma *R-therm-ctrl*:
Ref ⌈*I Tmin Tmax*⌉ ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ≥
(*2* ::= (λ*s*. *0*));
(*3* ::= (λ*s*. *s*$*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)
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* ⌈*I Tmin Tmax*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*LOOP*
Ref ⌈*I Tmin Tmax*⌉ ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉;
Ref ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*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 *Ref* ⌈*I Tmin Tmax*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*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*´= (λ*t*. *f a 0*) & *G Tmin Tmax a 0 on* (λ*s*. {*0*..τ}) *UNIV* @ *0*)
ELSE
(*x*´= (λ*t*. *f a L*) & *G Tmin Tmax a L on* (λ*s*. {*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 $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

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 $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$
$hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

— Verified with the flow

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

lemma *tank-arith*:
assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau \in \{0..\tau\}.\ \tau \leq - ((hmin - y) / c_o) \Longrightarrow hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0..\tau\}.\ \tau \leq (hmax - y) / (c_i - c_o) \Longrightarrow (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) \cdot \tau + y$
and $y \leq hmax \Longrightarrow y - c_o \cdot \tau \leq hmax$
apply(*simp-all add*: *field-simps le-divide-eq assms*)
using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

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

lemma *tank-flow*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *Hoare* ⌈*I hmin hmax*⌉
(*LOOP*
— control
$((2 ::=(\lambda s.0));(3 ::=(\lambda s.\ s\$1));$
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq hmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq hmax - 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x' = (\lambda t.\ f\ (c_i - c_o))\ \&\ G\ hmax\ (c_i - c_o)\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0)$
$ELSE\ (x' = (\lambda t.\ f\ (-c_o))\ \&\ G\ hmin\ (-c_o)\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0))\ )$
INV I hmin hmax) ⌈*I hmin hmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R*=$\lambda s.\ I\ hmin\ hmax\ s \wedge s\$2{=}0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.\ I\ hmin\ hmax\ s \wedge s\$2{=}0 \wedge s\$3 = s\$1$ **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.\ I\ hmin\ hmax\ s \wedge s\$2{=}0$ **in** *H-seq*, *simp*, *simp*)
apply(*rule H-cond*, *simp-all add*: *H-g-ode-tank*[*OF assms*(*1*)])
using *assms tank-arith*[*OF - assms*(*2*,*3*)] **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:

$0 \le \tau \Longrightarrow$ *diff-invariant* (*dI hmin hmax k*) ($\lambda t.$ *f k*) ($\lambda s.$ $\{0..\tau\}$) *UNIV 0 Guard*
apply(*intro diff-invariant-conj-rule*)
apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
apply(*rule-tac* ν'=$\lambda t.$ *0* **and** μ'=$\lambda t.$ *1* **in** *diff-invariant-leq-rule*, *simp-all*, *presburger*)
apply(*rule-tac* ν'=$\lambda t.$ *0* **and** μ'=$\lambda t.$ *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 $0 \le (\tau{::}real)$ **and** $c_o < c_i$ **and** *b*: $hmin \le y_0$ **and** *g*: $\tau \le (hmax - y_0) / (c_i - c_o)$
shows $hmin \le (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \le hmax$
proof−
have $(c_i - c_o) \cdot \tau \le (hmax - y_0)$
using *g assms*(*2*,*3*) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
thus $(c_i - c_o) \cdot \tau + y_0 \le hmax$
by *auto*
show $hmin \le (c_i - c_o) \cdot \tau + y_0$
using *b assms*(*1*,*2*) **by** (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)
qed

lemma *tank-inv-arith2*:
assumes $0 \le (\tau{::}real)$ **and** $0 < c_o$ **and** *b*: $y_0 \le hmax$ **and** *g*: $\tau \le -((hmin - y_0) / c_o)$
shows $hmin \le y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \le hmax$
proof−
have $\tau \cdot c_o \le y_0 - hmin$
using *g* ‹$0 < c_o$› *pos-le-minus-divide-eq* **by** *fastforce*
thus $hmin \le y_0 - c_o \cdot \tau$
by (*auto simp*: *mult.commute*)
show $y_0 - c_o \cdot \tau \le hmax$
using *b assms*(*1*,*2*) **by** (*smt mult-nonneg-nonneg*)
qed

lemma *tank-inv*:
assumes $0 \le \tau$ **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* ∧ *s\$3* ≤ *hmin* + *1*) *THEN* (*4* ::= ($\lambda s.1$)) *ELSE*
(*IF* ($\lambda s.$ *s\$4* = *1* ∧ *s\$3* ≥ *hmax* − *1*) *THEN* (*4* ::= ($\lambda s.0$)) *ELSE skip*));
— dynamics
(*IF* ($\lambda s.$ *s\$4* = *0*) *THEN*
(*x*´= ($\lambda t.$ *f* $(c_i{-}c_o)$) & *G hmax* $(c_i{-}c_o)$ *on* ($\lambda s.$ $\{0..\tau\}$) *UNIV* @ *0 DINV* (*dI hmin hmax* $(c_i{-}c_o)$)))
ELSE
(*x*´= ($\lambda t.$ *f* $(-c_o)$) & *G hmin* $(-c_o)$ *on* ($\lambda s.$ $\{0..\tau\}$) *UNIV* @ *0 DINV* (*dI hmin hmax* $(-c_o)$)))))
INV I hmin hmax) ⌈*I hmin hmax*⌉
apply(*rule H-loopI*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* ∧ *s\$2*=*0* ∧ *s\$3* = *s\$1* **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.$ *I hmin hmax s* ∧ *s\$2*=*0* ∧ *s\$3* = *s\$1* **in** *H-seq*)
apply(*rule-tac R*=$\lambda s.$ *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* ::=(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*´= (λ*t*. *f* (c_i−c_o)) & *G hmax* (c_i−c_o) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* (c_i−c_o)))
ELSE
(*x*´= (λ*t*. *f* (−c_o)) & *G hmin* (−c_o) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* (−c_o)))))
INV I hmin hmax) (**is** *LOOP* (*?ctrl*;*?dyn*) *INV* - ≤ *?ref*)
proof−
— First we refine the control.
let *?Icntrl* = λ*s*. *I hmin hmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*
and *?cond* = λ*s*. *s*$*4* = *0* ∧ *s*$*3* ≤ *hmin* + *1*
have *ifbranch1*: *4* ::= (λ*s*.*1*) ≤ *Ref* ⌈λ*s*. *?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch1*)
by (*rule R-assign-rule*, *simp*)
have *ifbranch2*: (*IF* (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*) ≤
Ref ⌈λ*s*. ¬ *?cond s* ∧ *?Icntrl s*⌉ ⌈*?Icntrl*⌉ (**is** - ≤ *?branch2*)
apply(*rule order-trans*, *rule R-cond-mono*) **defer defer**
by (*rule R-cond*) (*auto intro*!: *R-assign-rule R-skip*)
have *ifthenelse*: (*IF ?cond THEN ?branch1 ELSE ?branch2*) ≤ *Ref* ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉ (**is** *?ifthenelse* ≤ -)
by (*rule R-cond*)
have (*IF ?cond THEN* (*4* ::= (λ*s*.*1*)) *ELSE* (*IF* (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*)) ≤
Ref ⌈*?Icntrl*⌉ ⌈*?Icntrl*⌉
apply(*rule-tac y*=*?ifthenelse* **in** *order-trans*, *rule R-cond-mono*)
using *ifbranch1 ifbranch2 ifthenelse* **by** *auto*
hence *ctrl*: *?ctrl* ≤ *Ref* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉
apply(*rule-tac R*=*?Icntrl* **in** *R-seq-rule*)
apply(*rule-tac R*=λ*s*. *I hmin hmax s* ∧ *s*$*2* = *0* **in** *R-seq-rule*)
by (*auto intro*!: *R-assign-rule*)
— Then we refine the dynamics.
have *dynup*: (*x*´= (λ*t*. *f* (c_i−c_o)) & *G hmax* (c_i−c_o) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* (c_i−c_o))) ≤
Ref ⌈λ*s*. *s*$*4* = *0* ∧ *?Icntrl s*⌉ ⌈*I hmin hmax*⌉
apply(*rule R-g-ode-inv*[*OF tank-diff-inv*[*OF assms*(*1*)]])
using *assms* **by** (*auto simp*: *tank-inv-arith1*)
have *dyndown*: (*x*´= (λ*t*. *f* (−c_o)) & *G hmin* (−c_o) *on* (λ*s*. {*0*..τ}) *UNIV* @ *0 DINV* (*dI hmin hmax* (−c_o))) ≤
Ref ⌈λ*s*. *s*$*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* ≤ *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*⌉ ≤ ⌈*I hmin hmax*⌉
by *simp*
have *inv-inv*: *Ref* ⌈*I hmin hmax*⌉ ⌈*?Icntrl*⌉; (*Ref* ⌈*?Icntrl*⌉ ⌈*I hmin hmax*⌉) ≤ *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* ≤ *?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

0.11 Mathematical Preliminaries

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

theory *MTX-Preliminaries*
  imports *../HS-Preliminaries*

begin

0.11.1 Syntax

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

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

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

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

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

0.11.2 Topology and sets

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

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

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

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

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

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

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

lemma *finite-image-of-finite*[*simp*]:
  fixes *f*::*'a*::*finite* $\Rightarrow$ *'b*

shows $\mathit{finite}\ \{x.\ \exists\, i.\ x = f\ i\}$
using *finite-Atleast-Atmost-nat* **by** *force*

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

0.11.3 Functions

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

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

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

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

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

0.11.4 Suprema

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

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

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

lemma *cSup-finite-ex*:

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

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

lemma *finite-nat-minimal-witness*:
fixes *P* :: ('*a*::*finite*) $\Rightarrow$ *nat* $\Rightarrow$ *bool*
assumes $\forall$ *i*. $\exists$ *N*::*nat*. $\forall$ *n* $\geq$ *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*
assume *n* $\geq$ *?N*
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* λ*N*. $\forall$ *n* $\geq$ *N*. *P i n*] **by** *blast*
have *finite* {*?bound i* |*i*. *i* $\in$ *UNIV*}
by *simp*
hence *?N* $\geq$ *?bound i*
using *Max-ge* **by** *blast*
hence *n* $\geq$ *?bound i*
using ‹*n* $\geq$ *?N*› **by** *linarith*
hence *P i n*
using *obs* **by** *blast*}
thus $\exists$ *N*. $\forall$ *i n*. *N* $\leq$ *n* $\longrightarrow$ *P i n*
by *blast*
qed

0.11.5 Real numbers

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

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

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

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

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

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

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

0.11.6 Vectors and matrices

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

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

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

lemma *matrix-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*)
let $?\delta_K = \lambda i\ j\ k.$ *if* $i = j$ *then* k *else* 0
have ($\sum j{\in}UNIV.\ (\|(?\delta_K\ j\ i\ k)\|)^2$) = ($\sum j{\in}\{i\}.\ (\|(?\delta_K\ j\ i\ k)\|)^2$) + ($\sum j{\in}(UNIV-\{i\}).\ (\|(?\delta_K\ j\ i\ k)\|)^2$)
using *finite-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

0.11.7 Diagonalization

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

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

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

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

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

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

lemma *mat-inverse-simps*[*simp*]:
  fixes c :: 'a::division-ring
  assumes $c \neq 0$
  shows $mat\ (inverse\ c) ** mat\ c = mat\ 1$

and *mat c* $**$ *mat* (*inverse c*) = *mat 1*
unfolding *matrix-matrix-mult-def mat-def* **by** (*auto simp*: *vec-eq-iff assms*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *compow-matrix-mul-diag-mat-eq*: $((**)\ (diag\text{-}mat\ f)\ \hat{}\hat{}\ n)\ (mat\ 1) = diag\text{-}mat\ (\lambda i.\ f\ i\hat{}n)$
  apply(*induct n*, *simp-all add*: *matrix-mul-diag-matl*)
  by (*auto simp*: *vec-eq-iff diag-mat-def*)

lemma *compow-similar-diag-mat-eq*:
  assumes *invertible P*
    and $A = P^{-1}\ **\ (diag\text{-}mat\ f)\ **\ P$
  shows $((**)\ A\ \hat{}\hat{}\ n)\ (mat\ 1) = P^{-1}\ **\ (diag\text{-}mat\ (\lambda i.\ f\ i\hat{}n))\ **\ P$
proof(*induct n*, *simp-all add*: *assms*)
  fix *n*::*nat*
  have $P^{-1}\ **\ diag\text{-}mat\ f\ **\ P\ **\ (P^{-1}\ **\ diag\text{-}mat\ (\lambda i.\ f\ i\ \hat{}\ n)\ **\ P) =$
  $P^{-1}\ **\ diag\text{-}mat\ f\ **\ diag\text{-}mat\ (\lambda i.\ f\ i\ \hat{}\ n)\ **\ P$ (**is** *?lhs* = -)
    by (*metis* (*no-types*, *lifting*) *assms*(*1*) *invertibleD*(*2*) *matrix-mul-rid matrix-mul-assoc*)
  also have ... $= P^{-1}\ **\ diag\text{-}mat\ (\lambda i.\ f\ i * f\ i\ \hat{}\ n)\ **\ P$ (**is** - = *?rhs*)
    by (*metis* (*full-types*) *matrix-mul-assoc matrix-mul-diag-diag*)
  finally show *?lhs* = *?rhs* **.**
qed

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

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

end

0.12 Matrix norms

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

theory *MTX-Norms*
  imports *MTX-Preliminaries*

begin

0.12.1 Matrix operator norm

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

lemma *norm-matrix-bound*:
  fixes A :: $('a{::}real\text{-}normed\text{-}algebra\text{-}1)\,\hat{}\,'n\,\hat{}\,'m$
  shows $\|x\| = 1 \Longrightarrow \|A *v\ x\| \le \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
proof−
  fix x :: $('a,\ 'n)\ vec$ **assume** $\|x\| = 1$
  hence *xi-le1*: $\bigwedge i.\ \|x\ \$\ i\| \le 1$
    by (*metis Finite-Cartesian-Product.norm-nth-le*)
  {**fix** $j{::}'m$
    have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \le (\sum i{\in}UNIV.\ \|A\ \$\ j\ \$\ i * x\ \$\ i\|)$
      using *norm-sum* **by** *blast*
    also have ... $\le (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * (\|x\ \$\ i\|))$
      by (*simp add*: *norm-mult-ineq sum-mono*)
    also have ... $\le (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$
      using *xi-le1* **by** (*simp add*: *sum-mono mult-left-le*)
    finally have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \le (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$ **by** *simp*}
  hence $\bigwedge j.\ \|(A *v\ x)\ \$\ j\| \le ((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j$
    unfolding *matrix-vector-mult-def* **by** *simp*
  hence $(\sum j{\in}UNIV.\ (\|(A *v\ x)\ \$\ j\|)^2) \le (\sum j{\in}UNIV.\ (\|((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j\|)^2)$
    by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)
  thus $\|A *v\ x\| \le \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
    unfolding *norm-vec-def L2-set-def* **by** *simp*
qed

lemma *onorm-set-proptys*:
  fixes A :: $('a{::}real\text{-}normed\text{-}algebra\text{-}1)\,\hat{}\,'n\,\hat{}\,'m$
  shows *bounded* (*range* $(\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|))$)
    and *bdd-above* (*range* $(\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|))$)
    and (*range* $(\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|))) \ne \{\}$
  unfolding *bounded-def bdd-above-def image-def dist-real-def*
    apply(*rule-tac* $x=0$ **in** *exI*)
  by (*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*)+ *force*

lemma *op-norm-set-proptys*:
  fixes A :: $('a{::}real\text{-}normed\text{-}algebra\text{-}1)\,\hat{}\,'n\,\hat{}\,'m$
  shows *bounded* $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
    and *bdd-above* $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
    and $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\} \ne \{\}$
  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*: $\|A\|_{op} = Sup\ \{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
  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*: $0 \le \|A\|_{op}$
  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*: $\|A *v\ sgn\ x\| \le \|A\|_{op}$
  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*: $\|A *v\ x\| \le (\|A\|_{op}) * (\|x\|)$
proof−
  have $\|A *v\ x\| = (\|A *v\ sgn\ x\|) * (\|x\|)$
    by(*simp add*: *mult-norm-matrix-sgn-eq*)
  also have ... $\le (\|A\|_{op}) * (\|x\|)$
    using *norm-sgn-le-op-norm*[*of A*] **by** (*simp add*: *mult-mono′*)
  finally show *?thesis* **by** *simp*
qed

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

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

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

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

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

lemma *op-norm-matrix-matrix-mult-le*: $\|A ** B\|_{op} \le (\|A\|_{op}) * (\|B\|_{op})$
proof(*rule onorm-le*)
  have $0 \le (\|A\|_{op})$
    by(*rule onorm-pos-le*[*OF blin-matrix-vector-mult*])
  fix *x* **have** $\|A ** B *v\ x\| = \|A *v\ (B *v\ x)\|$
    by (*simp add*: *matrix-vector-mul-assoc*)
  also have ... $\le (\|A\|_{op}) * (\|B *v\ x\|)$
    by (*simp add*: *norm-matrix-le-mult-op-norm*[*of - B ∗v x*])
  also have ... $\le (\|A\|_{op}) * ((\|B\|_{op}) * (\|x\|))$
    using *norm-matrix-le-mult-op-norm*[*of B x*] ‹$0 \le (\|A\|_{op})$› *mult-left-mono* **by** *blast*
  finally show $\|A ** B *v\ x\| \le (\|A\|_{op}) * (\|B\|_{op}) * (\|x\|)$
    by *simp*
qed

lemma *norm-matrix-vec-mult-le-transpose*:
  $\|x\| = 1 \Longrightarrow (\|A *v\ x\|) \le sqrt\ (\|transpose\ A ** A\|_{op}) * (\|x\|)$ **for** *A* :: *real*^′*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 ... $\le (\|x\|) * (\|transpose\ A *v\ (A *v\ x)\|)$
    using *norm-cauchy-schwarz* **by** *blast*
  also have ... $\le (\|transpose\ A ** A\|_{op}) * (\|x\|)\,\hat{}\,2$
    apply(*subst matrix-vector-mul-assoc*)
    using *norm-matrix-le-mult-op-norm*[*of transpose A ∗∗ A x*]
    by (*simp add*: ‹$\|x\| = 1$›)

finally have $((\|A *v\ x\|))\hat{}2 \leq (\|transpose\ A ** A\|_{op}) * (\|x\|)\hat{}2$
  by *linarith*
thus $(\|A *v\ x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
  by (*simp add*: ‹$\|x\| = 1$› *real-le-rsqrt*)
qed

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

lemma *op-norm-le-transpose*: $\|A\|_{op} \leq \|transpose\ A\|_{op}$ **for** $A :: real\hat{}'n\hat{}'n$
proof−
  have *obs*:$\forall x.\ \|x\| = 1 \longrightarrow (\|A *v\ x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
    using *norm-matrix-vec-mult-le-transpose* **by** *blast*
  have $(\|A\|_{op}) \leq sqrt\ ((\|transpose\ A ** A\|_{op}))$
    using *obs* **apply**(*unfold op-norm-def*)
    by (*rule cSup-least*[*OF op-norm-set-proptys*(*3*)]) *clarsimp*
  hence $((\|A\|_{op}))^2 \leq (\|transpose\ A ** A\|_{op})$
    using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *op-norm-ge-0*
    by (*metis not-le real-less-lsqrt*)
  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** *metis*
qed

0.12.2 Matrix maximum norm

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

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

lemma *max-norm-set-proptys*: *finite* $\{|A\ \$\ i\ \$\ j| \mid i\ j.\ i \in UNIV \wedge j \in UNIV\}$ (**is** *finite ?X*)
proof−
  have $\bigwedge i.$ *finite* $\{|A\ \$\ i\ \$\ j| \mid j.\ j \in UNIV\}$
    using *finite-Atleast-Atmost-nat* **by** *fastforce*
  hence *finite* $(\bigcup i{\in}UNIV.\ \{|A\ \$\ i\ \$\ j| \mid j.\ j \in UNIV\})$ (**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*: $0 \le \|A\|_{max}$
  unfolding *max-norm-def*
  apply(*rule order.trans*[*OF abs-ge-zero*[*of A* \$ - \$ -] *Max-ge*])
  using *max-norm-set-proptys* **by** *auto*

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

lemma *sqrt-Sup-power2-eq-Sup-abs*:
  finite A $\Longrightarrow A \neq \{\} \Longrightarrow$ *sqrt* (*Sup* $\{(f\ i)^2\ |i.\ i \in A\}$) = *Sup* $\{|f\ i|\ |i.\ i \in A\}$
proof(*rule sym*)
  assume *assms*: *finite A* $A \neq \{\}$
  then obtain i **where** *i-def*: $i \in A \wedge$ *Sup* $\{(f\ i)^2|i.\ i \in A\} = (f\ i)$ˆ*2*
    using *cSup-finite-ex*[*of* $\{(f\ i)^2|i.\ i \in A\}$] **by** *auto*
  hence *lhs*: *sqrt* (*Sup* $\{(f\ i)^2\ |i.\ i \in A\}) = |f\ i|$
    by *simp*
  have *finite* $\{(f\ i)^2|i.\ i \in A\}$
    using *assms* **by** *simp*
  hence $\forall j{\in}A.\ (f\ j)^2 \le (f\ i)^2$
    using *i-def cSup-upper*[*of* - $\{(f\ i)^2\ |i.\ i \in A\}$] **by** *force*
  hence $\forall j{\in}A.\ |f\ j| \le |f\ i|$
    using *abs-le-square-iff* **by** *blast*
  also have $|f\ i| \in \{|f\ i|\ |i.\ i \in A\}$
    using *i-def* **by** *auto*
  ultimately show *Sup* $\{|f\ i|\ |i.\ i \in A\}$ = *sqrt* (*Sup* $\{(f\ i)^2\ |i.\ i \in A\}$)
    using *cSup-mem-eq*[*of* $|f\ i|$ $\{|f\ i|\ |i.\ i \in A\}$] *lhs* **by** *auto*
qed

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

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

hence *2*: $\exists x \in \{\|$*diag-mat f* $*v\ x\| \mid x.\ \|x\| = 1\}$. *Max ?A* $\leq x$
by (*metis* (*mono-tags*, *lifting*) *abs-1 mem-Collect-eq norm-axis-eq order-refl real-norm-def*)
show *Sup* $\{\|$*diag-mat f* $*v\ x\| \mid x.\ \|x\| = 1\}$ = *Max ?A*
by (*rule cSup-eq*[*OF 1 2*])
qed

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

end

0.13 Square Matrices

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

theory *SQ-MTX*
imports *MTX-Norms*

begin

0.13.1 Definition

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

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

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

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

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

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

lift-definition *sq-mtx-diag* :: ((*'m::finite*) ⇒ *real*) ⇒ (*'m::finite*) *sq-mtx* (**binder** diag *10*)
is *diag-mat* .

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

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

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

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

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

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

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

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

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

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

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

lemma *sq-mtx-vec-mult-eq*: *m* $*_V$ *x* = (χ *i*. *sum* (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-mult-canon*: *A* $*_V$ (e *i*) = col *i A*
  by (*transfer*, *simp add*: *matrix-vector-mult-basis*)

0.13.2 Ring of square matrices

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

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

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

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

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

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

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

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

end

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

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

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

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

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

lemma *sq-mtx-minus-eq*: $A - B =$ *to-mtx* $(\chi\ i\ j.\ A$$i$j - B$$i$j)$
 by *transfer* (*simp add*: *vec-eq-iff*)

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

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

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

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

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

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

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

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

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

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

lemma *mtx-vec-mult-add-rdistr*: $(A + B) *_V x = A *_V x + B *_V x$

unfolding *plus-sq-mtx-def*
apply(*transfer*)
by (*simp add*: *matrix-vector-mult-add-rdistrib*)

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

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

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

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

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

0.13.3 Real normed vector space of square matrices

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

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

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

definition *sgn-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx*
where $sgn\text{-}sq\text{-}mtx\ A = (inverse\ (\|A\|)) *_R A$

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

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

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

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

end

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

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

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

by (*unfold scaleR-sq-mtx-def*, *transfer*, *simp*)

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

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

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

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

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

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

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

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

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

lemma *norm-sq-mtx-diag*: $\|\textit{sq-mtx-diag}\ f\| = Max\ \{|f\ i| \,|i.\ i \in UNIV\}$
unfolding *norm-sq-mtx-def* **apply** *transfer*
by (*rule op-norm-diag-mat-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*: $\|A\ \$\$\ i\| \le \|A\|$
using *norm-vec-mult-le*[*of* $A^\dagger$ e i] **by** *simp*

0.13.4 Real normed algebra of square matrices

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

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

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

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

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

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

end

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

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

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

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

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

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

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

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

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

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

lemma *sq-mtx-inv-unique*:

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

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

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

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

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

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

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

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

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

lemma *power-sq-mtx-diag*: (*sq-mtx-diag* f)$\hat{\ }n$ = (diag i. $f\ i\hat{\ }n$)
by (*induct* n, *simp-all*)

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

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

0.13.5 Banach space of square matrices

```
lemma Cauchy-cols:
  fixes X :: nat ⇒ ('a::finite) sq-mtx
  assumes Cauchy X
  shows Cauchy (λn. col i (X n))
proof(unfold Cauchy-def dist-norm, clarsimp)
  fix ε::real assume ε > 0
  then obtain M where M-def:∀ m≥M. ∀ n≥M. ‖X m − X n‖ < ε
    using ‹Cauchy X› unfolding Cauchy-def by(simp add: dist-sq-mtx-def) metis
  {fix m n assume m ≥ M and n ≥ M
    hence ε > ‖X m − X n‖
      using M-def by blast
    moreover have ‖X m − X n‖ ≥ ‖(X m − X n) *V e i‖
      by(rule le-mtx-norm[of - X m − X n], force)
    moreover have ‖(X m − X n) *V e i‖ = ‖X m *V e i − X n *V e i‖
      by (simp add: mtx-vec-mult-minus-rdistrib)
    moreover have ... = ‖col i (X m) − col i (X n)‖
      by (simp add: mtx-vec-mult-minus-rdistrib mtx-vec-mult-canon)
    ultimately have ‖col i (X m) − col i (X n)‖ < ε
      by linarith}
  thus ∃ M. ∀ m≥M. ∀ n≥M. ‖col i (X m) − col i (X n)‖ < ε
    by blast
qed

lemma col-convergence:
  assumes ∀ i. (λn. col i (X n)) ⟶ L $ i
  shows X ⟶ to-mtx (transpose L)
proof(unfold LIMSEQ-def dist-norm, clarsimp)
  let ?L = to-mtx (transpose L)
  let ?a = CARD('a) fix ε::real assume ε > 0
  hence ε / ?a > 0 by simp
  hence ∀ i. ∃ N. ∀ n≥N. ‖col i (X n) − L $ i‖ < ε/?a
    using assms unfolding LIMSEQ-def dist-norm convergent-def by blast
  then obtain N where ∀ i. ∀ n≥N. ‖col i (X n) − L $ i‖ < ε/?a
    using finite-nat-minimal-witness[of λ i n. ‖col i (X n) − L $ i‖ < ε/?a] by blast
  also have ⋀i n. (col i (X n) − L $ i) = (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:∀ i. ∀ n≥N. ‖col i (X n − ?L)‖ < ε/?a
    by auto
  have ∀ n≥N. ‖X n − ?L‖ < ε
  proof(rule allI, rule impI)
    fix n::nat assume N ≤ n
    hence ∀ i. ‖col i (X n − ?L)‖ < ε/?a
      using N-def by blast
    hence (∑ i∈UNIV. ‖col i (X n − ?L)‖) < (∑ (i::'a)∈UNIV. ε/?a)
      using sum-strict-mono[of - λi. ‖col i (X n − ?L)‖] by force
    moreover have ‖X n − ?L‖ ≤ (∑ i∈UNIV. ‖col i (X n − ?L)‖)
      using sq-mtx-norm-le-sum-col by blast
    moreover have (∑ (i::'a)∈UNIV. ε/?a) = ε
      by force
    ultimately show ‖X n − ?L‖ < ε
      by linarith
  qed
  thus ∃ no. ∀ n≥no. ‖X n − ?L‖ < ε
    by blast
qed

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

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

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

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

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

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

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

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

0.13.6 Examples

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

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

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

2x2 matrices

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

lemma *mtx2-to-mtx*: *mtx*
  ([*a*, *b*] #
   [*c*, *d*] # []) =
  to-mtx (χ *i j*::*2*. *if i*=*1* ∧ *j*=*1 then a*
  else (*if i*=*1* ∧ *j*=*2 then b*
  else (*if i*=*2* ∧ *j*=*1 then c*
  else d)))
  apply(*subst sq-mtx-eq-iff*)
  using *exhaust-2* **by** *force*

abbreviation *diag2* :: *real* ⇒ *real* ⇒ *2 sq-mtx*
  where *diag2* ι_1 ι_2 ≡ *mtx*
   ([ι_1, *0*] #
    [*0*, ι_2] # [])

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

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

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

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

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

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

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

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

3x3 matrices

lemma *mtx3-to-mtx*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) =

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

abbreviation *diag3* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *3 sq-mtx*
where *diag3* ι_1 ι_2 ι_3 $\equiv$ *mtx*
([ι_1, *0*, *0*] #
[*0*, ι_2, *0*] #
[*0*, *0*, ι_3] # [])

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

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

lemma *zero-mtx3*: (*0*::*3 sq-mtx*) = *diag3 0 0 0*
by (*simp add*: *sq-mtx-eq-iff*)

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

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

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

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

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

end

0.14 Affine systems of ODEs

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

theory *MTX-Flows*
imports *SQ-MTX ../HS-ODEs*

begin

0.14.1 Existence and uniqueness for affine systems

definition *matrix-continuous-on* :: *real set* ⇒ (*real* ⇒ (*'a::real-normed-algebra-1*)^*'n*^*'m*) ⇒ *bool*
where *matrix-continuous-on T A* = (∀ *t* ∈ *T*. ∀ε > *0*. ∃ δ > *0*. ∀τ∈*T*. |τ − *t*| < δ ⟶ ‖*A* τ − *A t*‖$_{op}$ ≤ ε)

lemma *continuous-on-matrix-vector-multl*:
assumes *matrix-continuous-on T A*
shows *continuous-on T* (λ*t*. *A t* ∗*v s*)
proof(*rule continuous-onI*, *simp add*: *dist-norm*)
fix *e t*::*real* **assume** *0* < *e* **and** *t* ∈ *T*
let *?ε* = *e*/(‖(*if s* = *0 then 1 else s*)‖)
have *?ε* > *0*
using ‹*0* < *e*› **by** *simp*
then obtain δ **where** *dHyp*: δ > *0* ∧ (∀τ∈*T*. |τ − *t*| < δ ⟶ ‖*A* τ − *A t*‖$_{op}$ ≤ *?ε*)
using *assms* ‹*t* ∈ *T*› **unfolding** *dist-norm matrix-continuous-on-def* **by** *fastforce*
{**fix** τ **assume** τ ∈ *T* **and** |τ − *t*| < δ
have *obs*: *?ε* ∗ (‖*s*‖) = (*if s* = *0 then 0 else e*)
by *auto*
have ‖*A* τ ∗*v s* − *A t* ∗*v s*‖ = ‖(*A* τ − *A t*) ∗*v s*‖
by (*simp add*: *matrix-vector-mult-diff-rdistrib*)
also have ... ≤ (‖*A* τ − *A t*‖$_{op}$) ∗ (‖*s*‖)
using *norm-matrix-le-mult-op-norm* **by** *blast*
also have ... ≤ *?ε* ∗ (‖*s*‖)
using *dHyp* ‹τ ∈ *T*› ‹|τ − *t*| < δ› *mult-right-mono norm-ge-zero* **by** *blast*
finally have ‖*A* τ ∗*v s* − *A t* ∗*v s*‖ ≤ *e*
by (*subst* (*asm*) *obs*) (*metis* (*mono-tags*, *hide-lams*) ‹*0* < *e*› *less-eq-real-def order-trans*)}
thus ∃ *d*>*0*. ∀τ∈*T*. |τ − *t*| < *d* ⟶ ‖*A* τ ∗*v s* − *A t* ∗*v s*‖ ≤ *e*
using *dHyp* **by** *blast*

qed

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

lemma *local-lipschitz-affine*:
  fixes A :: *real* $\Rightarrow$ *'a::real-normed-algebra-1^'n^'m*
  assumes *open* T **and** *open* S
    and *Ahyp*: $\bigwedge\tau\ \varepsilon.\ \varepsilon > 0 \Longrightarrow \tau \in T \Longrightarrow$ *cball* $\tau\ \varepsilon \subseteq T \Longrightarrow$ *bdd-above* $\{\|A\ t\|_{op}\ |t.\ t \in$ *cball* $\tau\ \varepsilon\}$
  shows *local-lipschitz* $T\ S\ (\lambda t\ s.\ A\ t *v\ s + B\ t)$
proof(*unfold local-lipschitz-def lipschitz-on-def*, *clarsimp*)
  fix $s\ t$ **assume** $s \in S$ **and** $t \in T$
  then obtain *e1 e2* **where** *cball t e1* $\subseteq T$ **and** *cball s e2* $\subseteq S$ **and** *min e1 e2* > 0
    using *open-cballE*[*OF* - ‹*open T*›] *open-cballE*[*OF* - ‹*open S*›] **by** *force*
  hence *obs*: *cball t* (*min e1 e2*) $\subseteq T$
    by *auto*
  let *?L* = *Sup* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in$ *cball t* (*min e1 e2*)$\}$
  have $\|A\ t\|_{op} \in \{\|A\ \tau\|_{op}\ |\tau.\ \tau \in$ *cball t* (*min e1 e2*)$\}$
    using ‹*min e1 e2* > 0› **by** *auto*
  moreover have *bdd*: *bdd-above* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in$ *cball t* (*min e1 e2*)$\}$
    by (*rule Ahyp*, *simp only*: ‹*min e1 e2* > 0›, *simp-all add*: ‹$t \in T$› *obs*)
  moreover have *Sup* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in$ *cball t* (*min e1 e2*)$\} \geq 0$
    apply(*rule order.trans*[*OF op-norm-ge-0*[*of A t*]])
    by (*rule cSup-upper*[*OF calculation*])
  moreover have $\forall x\in$*cball s* (*min e1 e2*) $\cap S$. $\forall y\in$*cball s* (*min e1 e2*) $\cap S$.
    $\forall\tau\in$*cball t* (*min e1 e2*) $\cap T$. *dist* $(A\ \tau *v\ x)\ (A\ \tau *v\ y) \leq$ *?L* $*$ *dist x y*
    apply(*clarify*, *simp only*: *dist-norm*, *rule lipschitz-cond-affine*)
    using ‹*min e1 e2* > 0› *bdd* **by** *auto*
  ultimately show $\exists e>0.\ \exists L.\ \forall t\in$*cball t e* $\cap T$. $0 \leq L\ \wedge$
    $(\forall x\in$*cball s e* $\cap S$. $\forall y\in$*cball s e* $\cap S$. *dist* $(A\ t *v\ x)\ (A\ t *v\ y) \leq L *$ *dist x y*$)$
    using ‹*min e1 e2* > 0› **by** *blast*
qed

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

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

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

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

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

0.14.2 Flow for affine systems

Derivative rules for square matrices

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

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

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

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

lemma *vderiv-on-mtx-vec-multI*[*poly-derivatives*]:
assumes $D\ u = u'$ *on T* **and** $D\ A = A'$ *on T*
and $g = (\lambda t.\ A\ t *_V u'\ t + A'\ t *_V u\ t\ )$
shows $D\ (\lambda t.\ A\ t *_V u\ t) = g$ *on T*
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*

apply(*erule-tac x=x* **in** *ballE*, *simp-all*)+
apply(*rule derivative-eq-intros*(*142*))
by (*auto simp*: *fun-eq-iff mtx-vec-scaleR-commute pth-6 scaleR-mtx-vec-assoc*)

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

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

lemma *has-derivative-mtx-vec-multl*[*derivative-intros*]:
assumes $\bigwedge$ *i j*. *D* (λt. (*A t*) \$\$ *i* \$ *j*) $\mapsto$ ($\lambda\tau$. $\tau *_R$ (*A' t*) \$\$ *i* \$ *j*) (*at t within T*)
shows *D* (λt. *A t* $*_V$ *x*) $\mapsto$ ($\lambda\tau$. $\tau *_R$ (*A' t*) $*_V$ *x*) *at t within T*
unfolding *sq-mtx-vec-mult-sum-cols*
apply(*rule-tac f'1*=λi τ. $\tau *_R$ (*x* \$ *i* $*_R$ col *i* (*A' t*)) **in** *derivative-eq-intros*(*10*))
apply(*simp-all add*: *scaleR-right.sum*)
apply(*rule-tac g'1*=$\lambda\tau$. $\tau *_R$ col *i* (*A' t*) **in** *derivative-eq-intros*(*4*), *simp-all add*: *mult.commute*)
using *assms* **unfolding** *sq-mtx-col-def column-def* **apply**(*transfer*, *simp*)
apply(*rule has-derivative-vec-lambda*)
by (*simp add*: *scaleR-vec-def*)

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

— Automatically generated derivative rules from this subsubsection

thm *derivative-eq-intros*(*140*,*141*,*142*,*143*)

Existence and uniqueness with square matrices

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

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

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

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

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

lemmas *sq-mtx-unique-sol-autonomous-affine* = *picard-lindeloef.ivp-unique-solution*[*OF*
picard-lindeloef-sq-mtx-affine[*OF*
continuous-on-const
continuous-on-const
UNIV-I is-interval-univ
open-UNIV open-UNIV]
UNIV-I - subset-UNIV]

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

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

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

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

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

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

end

0.15 Verification examples

theory *MTX-Examples*
imports *MTX-Flows ../HS-VC-Spartan*

begin

0.15.1 Examples

abbreviation *hoareT* :: ($'a \Rightarrow bool$) $\Rightarrow$ ($'a \Rightarrow 'a$ *set*) $\Rightarrow$ ($'a \Rightarrow bool$) $\Rightarrow$ *bool*
(*PRE- HP - POST -* [*85*,*85*]*85*) **where** *PRE P HP X POST Q* $\equiv$ ($P \leq |X] Q$)

Verification by uniqueness.

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

abbreviation *mtx-circ-flow* :: *real* $\Rightarrow$ *real*ˆ*2* $\Rightarrow$ *real*ˆ*2* (φ)
where φ *t s* $\equiv$ (χ *i*. *if i* = *1 then s*\$*1* ∗ *cos t* + *s*\$*2* ∗ *sin t else* − *s*\$*1* ∗ *sin t* + *s*\$*2* ∗ *cos t*)

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

lemma *mtx-circ*:
PRE(λs. $r^2 = (s \$ 1)^2 + (s \$ 2)^2$)
HP x´=($*_V$) *A* & *G*
POST (λs. $r^2 = (s \$ 1)^2 + (s \$ 2)^2$)
apply(*subst local-flow.fbox-g-ode-subset*[*OF local-flow-sq-mtx-linear*])
unfolding *mtx-circ-flow-eq* **by** *auto*

no-notation *mtx-circ* (A)
and *mtx-circ-flow* (φ)

Flow of diagonalisable matrix.

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

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

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

lemma *invertible-mtx-chB-hOsc*: $a \neq b \Longrightarrow$ *mtx-invertible* (*P a b*)
apply(*rule mtx-invertibleI*[*of* - $(P\ a\ b)^{-1}$])
apply(*unfold inv-mtx-chB-hOsc scaleR-mtx2 times-mtx2 one-mtx2*)

by (*subst sq-mtx-eq-iff*, *simp add*: *vector-def frac-diff-eq1*)+

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

lemma *mtx-hOsc-solution-eq*:
fixes *a b* :: *real*
defines $\iota_1 \equiv (b - sqrt\ (b^2+4*a))/2$ **and** $\iota_2 \equiv (b + sqrt\ (b^2+4*a))/2$
defines $\Phi\ t \equiv mtx$ (
$[\iota_2*exp(t*\iota_1) - \iota_1*exp(t*\iota_2),\quad exp(t*\iota_2)-exp(t*\iota_1)]\#$
$[a*exp(t*\iota_2) - a*exp(t*\iota_1),\ \iota_2*exp(t*\iota_2)-\iota_1*exp(t*\iota_1)]\#[])$
assumes $b^2 + a * 4 > 0$ **and** $a \neq 0$
shows $P\ (-\iota_2/a)\ (-\iota_1/a) * (\text{diag } i.\ exp\ (t * (if\ i{=}1\ then\ \iota_1\ else\ \iota_2))) * (P\ (-\iota_2/a)\ (-\iota_1/a))^{-1}$
$= (1/sqrt\ (b^2 + a * 4)) *_R (\Phi\ t)$
unfolding *assms* **apply**(*subst inv-mtx-chB-hOsc*)
using *assms* **apply**(*simp-all add*: *mtx-times-scaleR-commute*, *subst sq-mtx-eq-iff*)
unfolding *UNIV-2 sq-mtx-times-eq sq-mtx-scaleR-eq sq-mtx-uminus-eq* **apply**(*simp-all add*: *axis-def*)
by (*auto simp*: *field-simps*, *auto simp*: *field-power-simps*)+

lemma *local-flow-mtx-hOsc*:
fixes *a b*
defines $\iota_1 \equiv (b - sqrt\ (b\hat{}2+4*a))/2$ **and** $\iota_2 \equiv (b + sqrt\ (b\hat{}2+4*a))/2$
defines $\Phi\ t \equiv mtx$ (
$[\iota_2*exp(t*\iota_1) - \iota_1*exp(t*\iota_2),\quad exp(t*\iota_2)-exp(t*\iota_1)]\#$
$[a*exp(t*\iota_2) - a*exp(t*\iota_1),\ \iota_2*exp(t*\iota_2)-\iota_1*exp(t*\iota_1)]\#[])$
assumes $b^2 + a * 4 > 0$ **and** $a \neq 0$
shows *local-flow* $((*_V)\ (A\ a\ b))\ UNIV\ UNIV\ (\lambda t.\ (*_V)\ ((1/sqrt\ (b^2 + a * 4)) *_R \Phi\ t))$
unfolding *assms* **using** *local-flow-sq-mtx-linear*[*of A a b*] *assms*
apply(*subst* (*asm*) *exp-scaleR-diagonal2*[*OF invertible-mtx-chB-hOsc mtx-hOsc-diagonalizable*])
  apply(*simp*, *simp*, *simp*)
by (*subst* (*asm*) *mtx-hOsc-solution-eq*) *simp-all*

lemma *overdamped-door-arith*:
assumes $b^2 + a * 4 > 0$ **and** $a < 0$ **and** $b \leq 0$ **and** $t \geq 0$ **and** $s1 > 0$
shows $0 \leq ((b + sqrt\ (b^2 + 4 * a)) * exp\ (t * (b - sqrt\ (b^2 + 4 * a)) / 2) / 2 -$
$(b - sqrt\ (b^2 + 4 * a)) * exp\ (t * (b + sqrt\ (b^2 + 4 * a)) / 2) / 2) * s1 / sqrt\ (b^2 + a * 4)$
proof(*subst diff-divide-distrib*[*symmetric*], *simp*)
have *f0*: $s1 / (2 * sqrt\ (b^2 + a * 4)) > 0$ (**is** $s1/?c3 > 0$)
  using *assms*(*1*,*5*) **by** *simp*
have *f1*: $(b - sqrt\ (b^2 + 4 * a)) < (b + sqrt\ (b^2 + 4 * a))$ (**is** $?c2 < ?c1$)
  and *f2*: $(b + sqrt\ (b^2 + 4 * a)) < 0$
  using *sqrt-ge-absD*[*of b* $b^2 + 4 * a$] *assms* **by** (*force*, *linarith*)
hence *f3*: $exp\ (t * ?c2 / 2) \leq exp\ (t * ?c1 / 2)$ (**is** $exp\ ?t1 \leq exp\ ?t2$)
  unfolding *exp-le-cancel-iff*
  using *assms*(*4*) **by** (*case-tac t=0*, *simp-all*)
hence $?c2 * exp\ ?t2 \leq ?c2 * exp\ ?t1$
  using *f1 f2 real-mult-le-cancel-iff2*[*of* $-?c2\ exp\ ?t1\ exp\ ?t2$] **by** *linarith*
also have ... $< ?c1 * exp\ ?t1$
  using *f1* **by** *auto*
also have... $\leq ?c1 * exp\ ?t1$
  using *f1 f2* **by** *auto*
ultimately show $0 \leq (?c1 * exp\ ?t1 - ?c2 * exp\ ?t2) * s1 / ?c3$
  using *f0 f1 assms*(*5*) **by** *auto*

qed

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

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

Flow of non-diagonalisable matrix.

abbreviation *mtx-cnst-acc* :: *3 sq-mtx* (*K*)
where $K \equiv$ *mtx* (
[*0*,*1*,*0*] #
[*0*,*0*,*1*] #
[*0*,*0*,*0*] # [])

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

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

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

lemma *exp-mtx-cnst-acc-simps*:
exp $(t *_R K)$ \$\$ *1* \$ *1* = *1* *exp* $(t *_R K)$ \$\$ *1* \$ *2* = *t* *exp* $(t *_R K)$ \$\$ *1* \$ *3* = $t\hat{}2/2$
exp $(t *_R K)$ \$\$ *2* \$ *1* = *0* *exp* $(t *_R K)$ \$\$ *2* \$ *2* = *1* *exp* $(t *_R K)$ \$\$ *2* \$ *3* = *t*
exp $(t *_R K)$ \$\$ *3* \$ *1* = *0* *exp* $(t *_R K)$ \$\$ *3* \$ *2* = *0* *exp* $(t *_R K)$ \$\$ *3* \$ *3* = *1*
unfolding *exp-mtx-cnst-acc one-mtx3 pow2-scaleR-mtx-cnst-acc* **by** *simp-all*

lemma *exp-mtx-cnst-acc-vec-mult-eq*: *exp* $(t *_R K) *_V s$ =
vector [$s\$3 * t\hat{}2/2 + s\$2 * t + s\$1$, $s\$3 * t + s\$2$, $s\$3$]
apply(*subst exp-mtx-cnst-acc*, *subst pow2-scaleR-mtx-cnst-acc*)
apply(*simp add*: *sq-mtx-vec-mult-eq vector-def*)
unfolding *UNIV-3* **by** (*simp add*: *fun-eq-iff*)

lemma *local-flow-mtx-cnst-acc*:
local-flow $((*_V)\ K)$ *UNIV UNIV* $(\lambda t\ s.\ ((t *_R K)^2 /_R 2 + (t *_R K) + 1) *_V s)$
using *local-flow-sq-mtx-linear*[*of K*] **unfolding** *exp-mtx-cnst-acc* **.**

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

lemma *docking-station*:
  assumes $d > x_0$ **and** $v_0 > 0$
  shows $PRE\ (\lambda s.\ s\$1 = x_0 \wedge s\$2 = v_0)$
  $HP\ ((3 ::= (\lambda s.\ -(v_0\ \hat{}2/(2*(d-x_0))))); x' =(*_V)\ K\ \&\ G)$
  $POST\ (\lambda s.\ s\$2 = 0 \longleftrightarrow s\$1 = d)$
  apply(*clarsimp simp*: *le-fun-def local-flow.fbox-g-ode-subset*[*OF local-flow-sq-mtx-linear*[*of K*]])
  unfolding *exp-mtx-cnst-acc-vec-mult-eq* **using** *assms* **by** (*simp add*: *docking-station-arith*)

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

end

0.16 Examples

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

theory *ARCH2020-Examples*
  imports *HS-VC-MKA-rel ../Matrices/MTX-Flows*

begin

0.16.1 Basic

no-notation *Archimedean-Field.ceiling* (⌈-⌉)

Basic assignment

lemma $\lceil\lambda s.\ s\$1 \geq (0::real)\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))\ \lceil\lambda s.\ s\$1 \geq 1\rceil$
by *simp*

Overwrite assignment on some branches

lemma $\lceil\lambda s.\ s\$1 \geq (0::real)\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))$
$(wp\ ((1 ::= (\lambda s.\ s\$1 + 1)) \cup (2 ::= (\lambda s.\ s\$1 + 1)))\ \lceil\lambda s.\ s\$1 \geq 1\rceil)$
by (*simp add*: *rel-aka.fbox-add2*)

Overwrite assignment in loop

lemma $\lceil\lambda s.\ s\$1 \geq (0::real)\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))$
$(wp\ (LOOP\ (1 ::= (\lambda s.\ s\$1 + 1))\ INV\ (\lambda s.\ s\$1 \geq 1))\ \lceil\lambda s.\ s\$1 \geq 1\rceil)$
apply(*subst rel-aka.fbox-mult*[*symmetric*])
by (*rule wp-loopI-break*, *auto*)

Overwrite assignment in ODE

lemma $0 \leq t \Longrightarrow \lceil\lambda s.\ s\$1 \geq (0::real)\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))$
$(wp\ (x' = (\lambda t\ s.\ (\chi\ i.\ if\ i=1\ then\ 2\ else\ 0))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 \geq 1\rceil)$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $\varphi=\lambda t\ s.\ (\chi\ i.\ if\ i=1\ then\ 2*t+s\$1\ else\ s\$i)$ **and** $T=UNIV$])
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)
apply(*clarsimp*, *rule-tac* $x=1$ **in** *exI*)+
by (*auto intro*!: *poly-derivatives*)

Overwrite with nondeterministic assignment

lemma $\lceil\lambda s.\ s\$1 \geq (0::real)\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))\ (wp\ ((1 ::= ?);\lceil\lambda s.\ s\$1 \geq 1\rceil)\ \lceil\lambda s.\ s\$1 \geq 1\rceil)$
by (*simp add*: *wp-rel*, *auto simp*: *p2r-def*)

Tests and universal quantification

lemma $\lceil\lambda s::real\hat{\ }2.\ s\$1 \geq 0\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))$
$(wp\ ((\lceil\lambda s.\ s\$1 \geq 2\rceil;(1 ::= (\lambda s.\ s\$1 - 1))) \cup (\lceil\lambda s.\ \forall i.\ s\$i \geq 1 \longrightarrow s\$2 \geq 1\rceil;(1 ::= (\lambda s.\ s\$2))))$
$\lceil\lambda s.\ s\$1 \geq 1\rceil)$
by (*simp add*: *wp-rel*, *auto simp*: *p2r-def assign-def vec-upd-def*)

Overwrite assignment several times

lemma $0 \leq t \Longrightarrow \lceil\lambda s::real\hat{\ }2.\ s\$1 \geq 0 \wedge s\$2 \geq 1\rceil \leq wp\ (1 ::= (\lambda s.\ s\$1 + 1))$
$(wp\ ((LOOP\ (1 ::= (\lambda s.\ s\$1 + 1))\ INV\ (\lambda s.\ s\$1 \geq 1)) \cup (2 ::= (\lambda s.\ s\$1 + 1)))$
$(wp\ (x' = (\lambda t\ s.\ (\chi\ i.\ if\ i=2\ then\ 2\ else\ 0))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ (wp\ (1 ::= (\lambda s.\ s\$2))$
$\lceil\lambda s.\ s\$1 \geq 1\rceil)))$
apply(*simp*, *subst local-flow.wp-g-ode-ivl*[**where** $\varphi=\lambda t\ s.\ (\chi\ i.\ if\ i=2\ then\ 2*t+s\$2\ else\ s\$i)$ **and** $T=UNIV$])
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)
apply (*clarsimp*, *rule-tac* $x=1$ **in** *exI*, *force*)
apply(*auto intro*!: *poly-derivatives vec-eq-iff*)[*1*]
apply(*subst change-loopI*[**where** $I=\lambda s.\ 1 \leq s\$1 \wedge 1 \leq s\$2$])
apply(*subst rel-aka.fbox-mult*[*symmetric*])
apply(*rule rel-aka.fbox-seq-var*)
apply(*subst wp-assign*[**where** $Q=\lambda s.\ 1 \leq s\$1 \wedge 1 \leq s\$2$], *simp*)
apply(*subst le-wp-choice-iff*, *rule conjI*)
by (*subst wp-loopI*, *auto*)

Potentially overwrite dynamics

lemma $0 \leq t \Longrightarrow \lceil\lambda s::real\hat{\ }2.\ s\$1 > 0 \wedge s\$2 > 0\rceil \leq$
$wp\ (x' = (\lambda t\ s.\ (\chi\ i.\ if\ i=1\ then\ 5\ else\ 0))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)$
$(wp\ ((LOOP\ (1 ::= (\lambda s.\ s\$1 + 3))\ INV\ (\lambda s.\ 0 < s\$1)) \cup (2 ::= (\lambda s.\ s\$1)))$

```
⌈λs. s$1 > 0 ∧ s$2 > 0⌉)
  apply(subst rel-aka.fbox-mult[symmetric])+
  apply(rule rel-aka.fbox-seq-var)+
   apply(subst local-flow.wp-g-ode-ivl[where φ=λt s. (χ i. if i=1 then 5∗t+s$1 else s$i)
        and T=UNIV and Q=λs. s$1 > 0 ∧ s$2 > 0]; simp?)
   apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
    apply(clarsimp, rule-tac x=1 in exI)+
    apply (force, force intro!: poly-derivatives)
  apply(subst le-wp-choice-iff, rule conjI)
   apply(subst change-loopI[where I=λs. s$1 > 0 ∧ s$2 > 0])
  by (rule wp-loopI, simp-all)
```

Potentially overwrite exponential decay

```
abbreviation po-exp-dec-f :: real^2 ⇒ real^2 (f)
  where f s ≡ (χ i. if i=1 then −s$1 else 0)

abbreviation po-exp-dec-flow :: real ⇒ real^2 ⇒ real^2 (φ)
  where φ t s ≡ (χ i. if i=1 then s$1 ∗ exp (− t) else s$i)

lemma local-flow-exp-flow: local-flow f UNIV UNIV φ
  apply(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def)
     apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  apply(unfold UNIV-2, simp)
  apply (metis power2-commute real-sqrt-ge-abs1)
  by (auto intro!: poly-derivatives simp: forall-2 vec-eq-iff)


lemma 0 ≤ t ⟹ ⌈λs::real^2. s$1 > 0 ∧ s$2 > 0⌉ ≤ wp (x´= (λt. f) & G on (λs. {0..t}) UNIV @ 0)
  (wp ((LOOP (1 ::= (λs. s$1 + 3)) INV (λs. 0 < s$1)) ∪ (2 ::= (λs. s$1)))
  ⌈λs. s$1 > 0 ∧ s$2 > 0⌉)
 apply(subst rel-aka.fbox-mult[symmetric])+
  apply(rule rel-aka.fbox-seq-var)+
   apply(subst local-flow.wp-g-ode-ivl[OF local-flow-exp-flow, where Q=λs. s$1 > 0 ∧ s$2 > 0]; simp)
  apply(subst le-wp-choice-iff, rule conjI)
   apply(subst change-loopI[where I=λs. s$1 > 0 ∧ s$2 > 0])
  by (rule wp-loopI, auto)

no-notation po-exp-dec-f (f)
       and po-exp-dec-flow (φ)
```

Dynamics: Cascaded

```
lemma 0 ≤ t ⟹ ⌈λs::real^1. s$1 > 0⌉ ≤
  wp ((x´=(λt s. (χ i. if i=1 then 5 else 0)) & G on (λs. {0..t}) UNIV @ 0);
      (x´=(λt s. (χ i. if i=1 then 2 else 0)) & G on (λs. {0..t}) UNIV @ 0);
      (x´=(λt s. (χ i. if i=1 then s$1 else 0)) & G on (λs. {0..t}) UNIV @ 0))
  ⌈λs. s$1 > 0⌉
  apply(simp, subst local-flow.wp-g-ode-ivl[where T=UNIV and φ=λt s. (χ i. s$1 ∗ exp t)]; simp?)
   apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
    apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
    apply(force, force intro!: poly-derivatives)
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and φ=λt s. (χ i. 2∗t+s$1)]; simp?)
   apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
    apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
    apply(force, force intro!: poly-derivatives)
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and φ=λt s. (χ i. 5∗t+s$1)]; simp?)
  apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
    apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  apply(force, force intro!: poly-derivatives)
```

by *auto* (*smt exp-gt-zero mult-minus-left real-mult-less-iff1*)

Dynamics: Single integrator time

lemma $0 \le t \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 = 0\rceil \le wp\ (x\acute{}{=}(\lambda t\ s.\ (\chi\ i.\ 1))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 \ge 0\rceil$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $T{=}UNIV$ **and** $\varphi{=}\lambda t\ s.\ (\chi\ i.\ t{+}s\$1)$]; *simp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac x=1* **in** *exI*)+
by (*auto intro*!: *poly-derivatives*)

Dynamics: Single integrator

lemma $0 \le t \Longrightarrow \lceil\lambda s{::}real\hat{\ }2.\ s\$1 \ge 0 \wedge s\$2 \ge 0\rceil \le$
$wp\ (x\acute{}{=}(\lambda t\ s.\ (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ 0))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 \ge 0\rceil$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $T{=}UNIV$ **and**
$\varphi{=}\lambda t\ s.\ (\chi\ i.\ if\ i = 1\ then\ s\$2{*}t{+}s\$1\ else\ s\$i)$]; *simp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*simp add*: *dist-norm norm-vec-def L2-set-def*)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac x=1* **in** *exI*)+
unfolding *UNIV-2* **by** (*auto intro*!: *poly-derivatives simp*: *forall-2 vec-eq-iff*)

Dynamics: Double integrator

lemma $0 \le t \Longrightarrow \lceil\lambda s{::}real\hat{\ }3.\ s\$1 \ge 0 \wedge s\$2 \ge 0 \wedge s\$3 \ge 0\rceil \le$
$wp\ (x\acute{}{=}(\lambda t\ s.\ (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ (if\ i = 2\ then\ s\$3\ else\ 0)))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)$
$\lceil\lambda s.\ s\$1 \ge 0\rceil$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $T{=}UNIV$ **and**
$\varphi{=}\lambda t\ s.\ (\chi\ i.\ if\ i = 1\ then\ s\$3{*}t^2/2 + s\$2{*}t + s\$1\ else\ (if\ i = 2\ then\ s\$3{*}t{+}s\$2\ else\ s\$i))$]; *simp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac x=1* **in** *exI*)+
unfolding *UNIV-3* **by** (*auto intro*!: *poly-derivatives simp*: *forall-3 vec-eq-iff*)

Dynamics: Triple integrator

abbreviation *triple-int-f* :: $real\hat{\ }4 \Rightarrow real\hat{\ }4\ (f)$
where $f\ s \equiv (\chi\ i.\ if\ i = 1\ then\ s\$2\ else\ (if\ i = 2\ then\ s\$3\ else\ (if\ i = 3\ then\ s\$4\ else\ (s\$4)^2)))$

lemma $0 \le t \Longrightarrow \lceil\lambda s{::}real\hat{\ }4.\ s\$1 \ge 0 \wedge s\$2 \ge 0 \wedge s\$3 \ge 0 \wedge s\$4 \ge 0\rceil \le$
$wp\ (x\acute{}{=}(\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)$
$\lceil\lambda s.\ s\$1 \ge 0\rceil$
apply(*rule-tac* $C{=}\lambda s.\ s\$4 \ge 0$ **in** *diff-cut-rule*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$4 \ge 0$])
apply(*rule wp-g-odei*; *simp?*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ (s\$4)^2$ **in** *diff-invariant-leq-rule*; *simp?*)
apply(*force intro*!: *poly-derivatives simp*: *forall-4*)
apply(*rule-tac* $C{=}\lambda s.\ s\$3 \ge 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$3 \ge 0$])
apply(*rule wp-g-odei*; *simp?*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ (s\$4)$ **in** *diff-invariant-rules*(2); *simp?*)
apply(*force intro*!: *poly-derivatives simp*: *forall-4*)
apply(*rule-tac* $C{=}\lambda s.\ s\$2 \ge 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$2 \ge 0$])
apply(*rule wp-g-odei*; *simp?*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ (s\$3)$ **in** *diff-invariant-rules*(2); *simp?*)
apply(*force intro*!: *poly-derivatives simp*: *forall-4*)
apply(*rule-tac* $C{=}\lambda s.\ s\$1 \ge 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$1 \ge 0$])
apply(*rule wp-g-odei*; *simp?*)

apply(*rule-tac* $\nu'=\lambda s.\ 0$ **and** $\mu'=\lambda s.\ (s\$2)$ **in** *diff-invariant-rules*(*2*); *simp?*)
apply(*force intro*!: *poly-derivatives simp*: *forall-4*)
by (*rule diff-weak-rule*, *simp*)

no-notation *triple-int-f* (*f*)

Dynamics: Exponential decay (1)

lemma $0 \leq t \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 > 0\rceil \leq wp\ (x\acute{}=(\lambda s.\ (\chi\ i.\ -\ s\$1))\ \&\ G)\ \lceil\lambda s.\ s\$1 > 0\rceil$
apply(*subst local-flow.wp-g-ode-subset*[**where** $T=UNIV$ **and** $\varphi=\lambda t\ s.\ (\chi\ i.\ s\$1 * exp\ (-\ t))$]; *simp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac* $x=1$ **in** *exI*)+
by (*auto intro*!: *poly-derivatives*)

Dynamics: Exponential decay (2)

lemma $0 \leq t \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 > 0\rceil \leq wp\ (x\acute{}=(\lambda t\ s.\ (\chi\ i.\ -\ s\$1 + 1))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 > 0\rceil$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $T=UNIV$ **and** $\varphi=\lambda t\ s.\ (\chi\ i.\ 1 - exp\ (-\ t) + s\$1 * exp\ (-\ t))$]; *clarsimp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac* $x=1$ **in** *exI*)+
by (*auto intro*!: *poly-derivatives simp*: *field-simps*) (*smt exp-gt-zero mult-pos-pos one-less-exp-iff*)

Dynamics: Exponential decay (3)

lemma $0 \leq t \Longrightarrow y > 0 \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 > 0\rceil \leq$
$wp\ (x\acute{}=(\lambda t\ s.\ (\chi\ i.\ -\ y * s\$1))\ \&\ G\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 > 0\rceil$
apply(*subst local-flow.wp-g-ode*[**where** $T=UNIV$ **and** $\varphi=\lambda t\ s.\ (\chi\ i.\ s\$1 * (exp\ (-t * y)))$]; *clarsimp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp*, *rule-tac* $x=1$ **in** *exI*)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac* $x=y$ **in** *exI*)
apply (*metis abs-mult abs-of-pos dist-commute dist-real-def less-eq-real-def vector-space-over-itself.scale-right-diff-distrib*)
by (*auto intro*!: *poly-derivatives simp*: *field-simps*)

Dynamics: Exponential growth (1)

lemma $0 \leq t \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 \geq 0\rceil \leq$
$wp\ (x\acute{}=(\lambda t\ s.\ (\chi\ i.\ s\$1))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 \geq 0\rceil$
apply(*subst local-flow.wp-g-ode-ivl*[**where** $T=UNIV$ **and** $\varphi=\lambda t\ s.\ (\chi\ i.\ s\$1 * exp\ t)$]; *clarsimp?*)
apply(*unfold-locales*; (*simp add*: *local-lipschitz-def lipschitz-on-def vec-eq-iff*)?)
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac* $x=1$ **in** *exI*)+
by (*auto intro*!: *poly-derivatives*)

Dynamics: Exponential growth (2)

lemma $\lceil\lambda s{::}real\hat{\ }2.\ s\$1 \geq 0 \wedge s\$2 \geq 0\rceil \leq$
$wp\ (x\acute{}=(\lambda t\ s.\ (\chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (s\$2)^2))\ \&\ G\ on\ (\lambda s.\ \{0..\})\ \{s.\ s\$2 < 0\}\ @\ 0)\ \lceil\lambda s.\ s\$1 \geq 0\rceil$
apply(*rule-tac* $C=\lambda s.\ s\$2 \geq 0$ **in** *diff-cut-rule*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I=\lambda s.\ s\$2 \geq 0$], *rule wp-g-odei*; *simp?*)
apply(*rule-tac* $\nu'=\lambda s.\ 0$ **and** $\mu'=\lambda s.\ (s\$2)\hat{\ }2$ **in** *diff-invariant-rules*(*2*); (*simp add*: *forall-2*)?)
apply(*rule-tac* $C=\lambda s.\ s\$1 \geq 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I=\lambda s.\ s\$1 \geq 0$])
apply(*rule wp-g-odei*; *simp?*)
apply(*rule-tac* $\nu'=\lambda s.\ 0$ **and** $\mu'=\lambda s.\ (s\$2)$ **in** *diff-invariant-rules*(*2*); (*simp add*: *forall-2*)?)
by (*rule diff-weak-rule*, *simp*)

Dynamics: Exponential growth (4)

lemma $0 \leq t \Longrightarrow \lceil\lambda s{::}real\hat{\ }1.\ s\$1 > 0\rceil \leq$
$wp\ (x\acute{}=(\lambda t\ s.\ (\chi\ i.\ (s\$1)^2))\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ UNIV\ @\ 0)\ \lceil\lambda s.\ s\$1 > 0\rceil$

apply(*subst g-ode-inv-def*[*symmetric*, **where** $I=\lambda s.\ s\$1 > 0$], *rule wp-g-odei*, *simp-all*)
by (*rule-tac* $\nu'=\lambda s.\ 0$ **and** $\mu'=\lambda s.\ (s\$1)^2$ **in** *diff-invariant-rules*(*3*); *simp?*)

Dynamics: Exponential growth (5)

lemma $0 \leq t \Longrightarrow \lceil\lambda s{::}real\hat{}1.\ s\$1 \geq 1\rceil \leq$
wp ($x'=(\lambda t\ s.\ (\chi\ i.\ (s\$1)^2 + 2 \cdot (s\$1)\hat{}4))$ & *G on* ($\lambda s.\ \{0..t\}$) *UNIV* @ *0*) $\lceil\lambda s.\ s\$1\hat{}3 \geq (s\$1)^2\rceil$
apply(*rule-tac* $C=\lambda s.\ s\$1 \geq 1$ **in** *diff-cut-rule*; *simp?*)
apply (*rule-tac* $\nu'=\lambda s.\ 0$ **and** $\mu'=\lambda s.\ (s\$1)^2 + 2 \cdot s\$1\ \hat{}\ 4$ **in** *diff-invariant-rules*(*2*); *simp?*)
apply(*force intro*!: *poly-derivatives*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I=\lambda s.\ s\$1\hat{}3 \geq (s\$1)^2$], *rule wp-g-odei*, *simp-all add*: *power-increasing*)
apply (*rule-tac* $\nu'=\lambda s.\ 2 * s\$1 * ((s\$1)^2 + 2 \cdot (s\$1)\ \hat{}\ 4)$
and $\mu'=\lambda s.\ 3 * (s\$1)^2 * ((s\$1)^2 + 2 \cdot (s\$1)\hat{}4)$ **in** *diff-invariant-rules*(*2*); *clarsimp?*)
apply(*auto intro*!: *poly-derivatives simp*: *field-simps semiring-normalization-rules*(*27*,*28*))
apply(*subgoal-tac* $X\ \tau\ \$\ 1 \geq 1$)
apply(*subgoal-tac* $2 + 4 \cdot (X\ \tau)\$1\ \hat{}\ 2 \leq 3 \cdot (X\ \tau)\$1 + 6 \cdot (X\ \tau)\$1\ \hat{}\ 3$)
apply (*smt One-nat-def numerals*(*1*) *one-le-power power.simps*(*2*) *power-0 power-add-numeral power-less-power-Suc power-one-right semiring-norm*(*2*) *semiring-norm*(*4*) *semiring-norm*(*5*))
apply (*smt numeral-One one-le-power power-add-numeral power-commutes power-le-one power-less-power-Suc power-one-right semiring-norm*(*5*))
by *simp*

Dynamics: Rotational dynamics (1)

lemma $\lceil\lambda s{::}real\hat{}2.\ (s\$1)^2 + (s\$2)^2 = 1\rceil \leq$ *wp* ($x'= (\lambda s.(\chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ - s\$2\ \textit{else}\ s\$1))$ & *G*)
$\lceil\lambda s.\ (s\$1)^2 + (s\$2)^2 = 1\rceil$
by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

Dynamics: Rotational dynamics (2)

abbreviation *rot-dyn2-mtx* :: *3 sq-mtx* (*A*)
where $A \equiv$ *mtx*
([*0*, −*1*, *0*] #
[*0*, *0*, *1*] #
[*0*, −*1*, *0*] # [])

abbreviation *rot-dyn2-f* :: $real\hat{}3 \Rightarrow real\hat{}3$ (*f*)
where $f\ s \equiv (\chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ - s\$2\ \textit{else}\ (\textit{if}\ i = 2\ \textit{then}\ s\$3\ \textit{else}\ - s\$2))$

lemma *rot-dyn2-f-linear*: $f\ s = A *_V s$
apply(*simp add*: *vec-eq-iff sq-mtx-vec-mult-eq*)
unfolding *UNIV-3* **using** *exhaust-3* **by** *force*

abbreviation *rot-dyn2-flow* :: $real \Rightarrow real\hat{}3 \Rightarrow real\hat{}3$ (φ)
where $\varphi\ t\ s \equiv (\chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ - s\$3 + s\$1 + s\$3 * \cos t - s\$2 * \sin t$
$\textit{else}\ (\textit{if}\ i = 2\ \textit{then}\ s\$3 * \sin t + s\$2 * \cos t\ \textit{else}\ s\$3 * \cos t - s\$2 * \sin t))$

lemma *mtx-circ-flow-eq*: $\exp (t *_R A) *_V s = \varphi\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-sq-mtx-linear*, *symmetric*, **where** $U=\lambda s.\ UNIV$])
apply(*simp-all*, *rule ivp-solsI*, *simp-all add*: *sq-mtx-vec-mult-eq vec-eq-iff*)
unfolding *UNIV-3* **using** *exhaust-3*
by (*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)+

lemma $\lceil\lambda s{::}real\hat{}3.\ (s\$1)^2 + (s\$2)^2 = 1 \wedge s\$3 = s\$1\rceil \leq$ *wp* ($x'= f$ & *G*)
$\lceil\lambda s.\ (s\$1)^2 + (s\$2)^2 = 1 \wedge s\$3 = s\$1\rceil$
apply(*subst rot-dyn2-f-linear*, *subst local-flow.wp-g-ode-subset*[*OF local-flow-sq-mtx-linear*])
unfolding *mtx-circ-flow-eq* **by** *auto*

no-notation *rot-dyn2-f* (f)
and *rot-dyn2-mtx* (A)
and *rot-dyn2-flow* (φ)

Dynamics: Rotational dynamics (3)

lemma $\lceil\lambda s{::}real\hat{}4.\ (s\$3)^2 + (s\$4)^2 = w^2 \cdot p^2 \wedge s\$3 = -\ w \cdot s\$2 \wedge s\$4 = w \cdot s\$1\rceil \leq$
wp $(x\acute{} = (\lambda s.\ (\chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$3\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ s\$4\ \mathit{else}\ (\mathit{if}\ i = 3\ \mathit{then}\ -\ w * s\$4\ \mathit{else}\ w * s\$3)))) \ \&\ G)$
$\lceil\lambda s.\ (s\$3)^2 + (s\$4)^2 = w^2{*}p^2 \wedge s\$3 = -\ w * s\$2 \wedge s\$4{=}\ w * s\$1\rceil$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

Dynamics: Spiral to equilibrium

lemma $\lceil\lambda s{::}real\hat{}3.\ (s\$3) \geq 0 \wedge s\$1{=}0 \wedge s\$2{=}3\rceil \leq$
wp $(x\acute{} = (\lambda t\ s.\ (\chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ -\ (s\$3)^2{*}(s\$1)\ -\ 2{*}(s\$3){*}(s\$2)\ \mathit{else}\ 0)))\ \&\ G\ \mathit{on}\ (\lambda s.\ \{0..\})\ \mathit{UNIV}\ @\ 0)$
$\lceil\lambda s.\ (s\$3)^2{*}(s\$1)^2 + (s\$2)^2 \leq 9\rceil$
apply(*rule-tac* $C{=}\lambda s.\ s\$3 \geq 0$ **in** *diff-cut-rule*; *simp?*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$3 \geq 0$], *rule wp-g-odei*, *simp-all*)
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(*2*); (*simp add*: *forall-3*)*?*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ (s\$3)^2{*}(s\$1)^2 + (s\$2)^2 \leq 9$], *rule wp-g-odei*, *simp-all add*: *power-increasing*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 2 * (s\$3)^2 * (s\$1) * (s\$2) + 2 * (s\$2) * (-\ (s\$3)^2{*}(s\$1)\ -\ 2{*}(s\$3){*}(s\$2))$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(*2*))
by (*auto intro*!: *poly-derivatives simp*: *forall-3 field-simps*) (*simp add*: *mult.assoc*[*symmetric*] *power2-eq-square*)

Dynamics: Open cases

lemma *has-vderiv-mono-test*:
assumes *T-hyp*: *is-interval* T
and *d-hyp*: $D\ f = f'\ \mathit{on}\ T$
and *xy-hyp*: $x{\in}T\ y{\in}T\ x \leq y$
shows $\forall x{\in}T.\ (0{::}real) \leq f'\ x \Longrightarrow f\ x \leq f\ y$
and $\forall x{\in}T.\ f'\ x \leq 0 \Longrightarrow f\ x \geq f\ y$
proof−
have $\{x..y\} \subseteq T$
using *T-hyp xy-hyp* **by** (*meson atLeastAtMost-iff mem-is-interval-1-I subsetI*)
hence $D\ f = f'\ \mathit{on}\ \{x..y\}$
using *has-vderiv-on-subset*[*OF d-hyp*(*1*)] **by** *blast*
hence $(\bigwedge t.\ x \leq t \Longrightarrow t \leq y \Longrightarrow D\ f \mapsto (\lambda\tau.\ \tau *_R f'\ t)\ \mathit{at}\ t\ \mathit{within}\ \{x..y\})$
unfolding *has-vderiv-on-def has-vector-derivative-def* **by** *auto*
then obtain c **where** *c-hyp*: $c \in \{x..y\} \wedge f\ y - f\ x = (y - x) *_R f'\ c$
using *mvt-very-simple*[*OF xy-hyp*(*3*), *of f* $(\lambda t\ \tau.\ \tau *_R f'\ t)$] **by** *blast*
hence *mvt-hyp*: $f\ x = f\ y - f'\ c * (y - x)$
by (*simp add*: *mult.commute*)
also have $\forall x{\in}T.\ 0 \leq f'\ x \Longrightarrow ... \leq f\ y$
using *xy-hyp d-hyp c-hyp* $\langle\{x..y\} \subseteq T\rangle$ **by** *auto*
finally show $\forall x{\in}T.\ 0 \leq f'\ x \Longrightarrow f\ x \leq f\ y$ **.**
have $\forall x{\in}T.\ f'\ x \leq 0 \Longrightarrow f\ y - f'\ c * (y - x) \geq f\ y$
using *xy-hyp d-hyp c-hyp* $\langle\{x..y\} \subseteq T\rangle$ **by** (*auto simp*: *mult-le-0-iff*)
thus $\forall x{\in}T.\ f'\ x \leq 0 \Longrightarrow f\ x \geq f\ y$
using *mvt-hyp* **by** *auto*
qed

lemma *continuous-on-ge-ball-ge*:
continuous-on $T\ f \Longrightarrow x \in T \Longrightarrow f\ x > (k{::}real) \Longrightarrow \exists\varepsilon{>}0.\ \forall y{\in}\mathit{ball}\ x\ \varepsilon \cap T.\ f\ y > k$
unfolding *continuous-on-iff* **apply**(*erule-tac* $x{=}x$ **in** *ballE*; *clarsimp?*)
apply(*erule-tac* $x{=}f\ x - k$ **in** *allE*, *clarsimp simp*: *dist-norm*)
apply(*rename-tac* δ, *rule-tac* $x{=}\delta$ **in** *exI*, *clarsimp*)

apply(*erule-tac* $x=y$ **in** *ballE*; *clarsimp?*)
by (*subst* (*asm*) *abs-le-eq*, *simp-all add*: *dist-commute*)

lemma *current-vderiv-ge-always-ge*:
fixes *c*::*real*
assumes *init*: $c < x\ t_0$ **and** *ode*: $D\ x = x'\ on\ \{t_0..\}$
and *dhyp*: $x' = (\lambda t.\ g\ (x\ t))\ \forall x{\geq}c.\ g\ x \geq 0$
shows $\forall t{\geq}t_0.\ x\ t > c$
proof−
have *cont*: *continuous-on* $\{t_0..\}$ x
using *vderiv-on-continuous-on*[*OF ode*] **.**
{assume $\exists t{\geq}t_0.\ x\ t \leq c$
hence *inf*: $\{t.\ t > t_0 \wedge x\ t \leq c\} \neq \{\}$ *bdd-below* $\{t.\ t > t_0 \wedge x\ t \leq c\}$
using *init less-eq-real-def* **unfolding** *bdd-below-def* **by** *force* (*rule-tac* $x{=}t_0$ **in** *exI*, *simp*)
define t_1 **where** *t1-hyp*: $t_1 = Inf\ \{t.\ t > t_0 \wedge x\ t \leq c\}$
hence $t_0 \leq t_1$
using *le-cInf-iff*[*OF inf*, *of* t_0] **by** *auto*
have $x\ t_1 \geq c$
proof−
{assume $x\ t_1 < c$
hence *obs*: $x\ t_1 \leq c\ x\ t_0 \geq c\ t_1 \neq t_0$
using *init* **by** *auto*
hence $t_1 > t_0$
using ‹$t_0 \leq t_1$› **by** *auto*
then obtain k **where** *k2-hyp*: $k \geq t_0 \wedge k \leq t_1 \wedge x\ k = c$
using *IVT2'*[*of* $\lambda t.\ x\ t$, *OF obs*(*1*,*2*) - *continuous-on-subset*[*OF cont*]] **by** *auto*
hence $t_0 < k\ k < t_1$
using ‹$x\ t_1 < c$› *less-eq-real-def init* **by** *auto*
also have $t_1 \leq k$
using *cInf-lower*[*OF* - *inf*(*2*)] *k2-hyp calculation* **unfolding** *t1-hyp* **by** *auto*
ultimately have *False*
by *simp***}**
thus $x\ t_1 \geq c$
by *fastforce*
qed
hence *obs*: $\forall t{\in}\{t_0..{<}t_1\}.\ x\ t > c$
proof−
{assume $\exists t{\in}\{t_0..{<}t_1\}.\ x\ t \leq c$
then obtain k **where** *k2-hyp*: $k \geq t_0 \wedge k < t_1 \wedge x\ k \leq c$
by *auto*
hence $k > t_0$
using ‹$x\ t_0 > c$› *less-eq-real-def* **by** *auto*
hence $t_1 \leq k$
using *cInf-lower*[*OF* - *inf*(*2*)] *k2-hyp* **unfolding** *t1-hyp* **by** *auto*
hence *False*
using *k2-hyp* **by** *auto***}**
thus $\forall t{\in}\{t_0..{<}t_1\}.\ x\ t > c$
by *force*
qed
hence $\forall t{\in}\{t_0..t_1\}.\ x'\ t \geq 0$
using ‹$x\ t_1 \geq c$› *dhyp*(*2*) *less-eq-real-def*
unfolding *dhyp* **by** (*metis atLeastAtMost-iff atLeastLessThan-iff*)
hence $x\ t_0 \leq x\ t_1$
apply(*rule-tac* $f{=}\lambda t.\ x\ t$ **and** $T{=}\{t_0..t_1\}$ **in** *has-vderiv-mono-test*(*1*); *clarsimp*)
using *has-vderiv-on-subset*[*OF ode*] **apply** *force*
using ‹$t_0 \leq t_1$› **by** (*auto simp*: *less-eq-real-def*)
hence $c < x\ t_1$
using ‹$x\ t_1 \geq c$› *init* **by** *auto*
then obtain ε **where** *eps-hyp*: $\varepsilon > 0 \wedge (\forall t{\in}ball\ t_1\ \varepsilon \cap \{t_0..\}.\ c < x\ t)$
using *continuous-on-ge-ball-ge*[*of* - $\lambda t.\ x\ t$, *OF cont* - ‹$c < x\ t_1$›] ‹$t_0 \leq t_1$› **by** *auto*

hence ∀ $t \in \{t_0..<t_1+\varepsilon\}$. $c < x\ t$
using *obs* ‹$t_0 \leq t_1$› *ball-eq-greaterThanLessThan* **by** *auto*
hence ∀ $t \in \{t.\ t > t_0 \wedge x\ t \leq c\}$. $t_1+\varepsilon \leq t$
by (*metis* (*mono-tags*, *lifting*) *atLeastLessThan-iff less-eq-real-def mem-Collect-eq not-le*)
hence $t_1+\varepsilon \leq t_1$
using *le-cInf-iff* [*OF inf*] **unfolding** *t1-hyp* **by** *simp*
hence *False*
using *eps-hyp* **by** *auto*}
thus ∀ $t \geq t_0$. $c < x\ t$
by *fastforce*
qed

lemma *0* ≤ *t* ⟹ ⌈λ*s*::*real*ˆ*2*. *s*\$*1*ˆ*3*>*5* ∧ *s*\$*2*>*2*⌉ ≤
wp (*x*´= (λ*t s*. (χ *i*. *if i*=*1 then s*\$*1*ˆ*3* + *s*\$*1*ˆ*4 else 5* ∗ *s*\$*2* + *s*\$*2*ˆ*2*)) & *G on* (λ*s*. {*0*..}) *UNIV* @ *0*)
⌈λ*s*. *s*\$*1*ˆ*3*>*5* ∧ *s*\$*2*>*2*⌉
apply(*simp*, *rule diff-invariant-rules*, *simp-all add*: *diff-invariant-eq ivp-sols-def forall-2*; *clarsimp*)
apply(*frule-tac x*=λ*t*. *X t* \$ *1* ˆ *3* **and** *g*=λ*t*. *3* ∗ *t*ˆ*2* + *3* ∗ (*root 3 t*)ˆ*5* **in** *current-vderiv-ge-always-ge*)
apply(*rule poly-derivatives*, *simp*, *assumption*, *simp*)
apply (*force simp*: *field-simps odd-real-root-power-cancel*, *force simp*: *add-nonneg-pos*, *force*)
apply(*frule-tac x*=λ*t*. *X t* \$ *2* **in** *current-vderiv-ge-always-ge*)
by (*force*, *force*, *force simp*: *add-nonneg-pos*, *simp*)

Dynamics: Closed cases

lemma *z* = − *2* ⟹ ⌈λ*s*::*real*ˆ*2*. *s*\$*1* ≥ *1* ∧ *s*\$*2* = *10*⌉ ≤
wp (*x*´= (λ*s*. (χ *i*. *if i*=*1 then s*\$*2 else z* + (*s*\$*2*)ˆ*2* − *s*\$*2*)) & (λ*s*. *s*\$*2* ≥ *0*))
⌈λ*s*. *s*\$*1* ≥ *1* ∧ *s*\$*2* ≥ *0*⌉
apply(*subst g-ode-inv-def* [*symmetric*, **where** *I*=λ*s*. *s*\$*1* ≥ *1* ∧ *s*\$*2* ≥ *0*], *rule wp-g-odei*, *simp-all*)
apply(*rule diff-invariant-rules*)
apply(*rule-tac* ν′=λ*s*. *0* **and** μ′=λ*s*. *s*\$*2* **in** *diff-invariant-rules*(*2*), *simp-all add*: *diff-invariant-eq*)
by (*force intro*!: *poly-derivatives*)

Dynamics: Conserved quantity

lemma *dyn-cons-qty-arith*: (*36*::*real*) · ($x1^2$ · (*x1* · *x2* ˆ *3*)) −
(− (*24* · ($x1^2$ · *x2*) · *x1* ˆ *3* · $(x2)^2$) − *12* · ($x1^2$ · *x2*) · *x1* · *x2* ˆ *4*) −
36 · ($x1^2$ · (*x2* · *x1*)) · $(x2)^2$ − *12* · ($x1^2$ · (*x1* · *x2* ˆ *5*)) = *24* · (*x1* ˆ *5* · *x2* ˆ *3*)
(**is** *?t1* − (− *?t2* − *?t3*) − *?t4* − *?t5* = *?t6*)
proof−
have *eq1*: *?t1* = *?t4*
by (*simp add*: *power2-eq-square power3-eq-cube*)
have *eq2*: − (− *?t2* − *?t3*) = (*?t6* + *?t3*)
by (*auto simp*: *field-simps semiring-normalization-rules*(*27*))
have *eq3*: *?t3* = *?t5*
by (*auto simp*: *field-simps semiring-normalization-rules*(*27*))
show *?t1* − (− *?t2* − *?t3*) − *?t4* − *?t5* = *?t6*
unfolding *eq1 eq2 eq3* **by** (*simp add*: *field-simps semiring-normalization-rules*(*27*))
qed

lemma *0* ≤ *t* ⟹ ⌈λ*s*::*real*ˆ*2*. (*s*\$*1*)ˆ*4*∗(*s*\$*2*)ˆ*2*+(*s*\$*1*)ˆ*2*∗(*s*\$*2*)ˆ*4*−*3*∗(*s*\$*1*)ˆ*2*∗(*s*\$*2*)ˆ*2* + *1* ≤ *c*⌉ ≤
wp (*x*´= (λ*t s*. (χ *i*. *if i*=*1 then 2*∗(*s*\$*1*)ˆ*4*∗(*s*\$*2*)+*4*∗(*s*\$*1*)ˆ*2*∗(*s*\$*2*)ˆ*3*−*6*∗(*s*\$*1*)ˆ*2*∗(*s*\$*2*)
else −*4*∗(*s*\$*1*)ˆ*3*∗(*s*\$*2*)ˆ*2*−*2*∗(*s*\$*1*)∗(*s*\$*2*)ˆ*4*+*6*∗(*s*\$*1*)∗(*s*\$*2*)ˆ*2*)) & *G on* (λ*s*. {*0*..*t*}) *UNIV* @ *0*)
⌈λ*s*. (*s*\$*1*)ˆ*4*∗(*s*\$*2*)ˆ*2*+(*s*\$*1*)ˆ*2*∗(*s*\$*2*)ˆ*4*−*3*∗(*s*\$*1*)ˆ*2*∗(*s*\$*2*)ˆ*2* + *1* ≤ *c*⌉
apply(*simp*, *rule-tac* μ′=λ*s*. *0* **and** ν′=λ*s*. *0* **in** *diff-invariant-rules*(*2*); *clarsimp simp*: *forall-2*)
apply(*intro poly-derivatives*; (*assumption*)*?*, (*rule poly-derivatives*)*?*)
apply *force*+
apply(*clarsimp simp*: *algebra-simps*(*17*,*18*,*19*,*20*) *semiring-normalization-rules*(*27*,*28*))
by (*auto simp*: *dyn-cons-qty-arith*)

Dynamics: Darboux equality

```
lemma mult-abs-right-mono: a < b ⟹ a ∗ |c| ≤ b ∗ |c| for c::real
  by (simp add: mult-right-mono)

lemma local-lipschitz-first-order-linear:
  fixes c::real ⇒ real
  assumes continuous-on T c
  shows local-lipschitz T UNIV (λt. (∗) (c t))
proof(unfold local-lipschitz-def lipschitz-on-def, clarsimp simp: dist-norm)
  fix x t::real assume t ∈ T
  then obtain δ where d-hyp: δ > 0 ∧ (∀τ∈T. |τ − t| < δ ⟶ |c τ − c t| < max 1 |c t|)
    using assms unfolding continuous-on-iff
    apply(erule-tac x=t in ballE, erule-tac x=max 1 (|c t|) in allE; clarsimp)
    by (metis dist-norm less-max-iff-disj real-norm-def zero-less-one)
  {fix τ x₁ x₂
    assume τ ∈ cball t (δ/2) ∩ T x₁ ∈ cball x (δ/2) x₂ ∈ cball x (δ/2)
    hence |τ − t| < δ τ ∈ T
      by (auto simp: dist-norm, smt d-hyp)
    hence |c τ − c t| < max 1 |c t|
      using d-hyp by auto
    hence − (max 1 |c t| + |c t|) < c τ ∧ c τ < max 1 |c t| + |c t|
      by (auto simp: abs-le-eq)
    hence obs: |c τ| < max 1 |c t| + |c t|
      by (simp add: abs-le-eq)
    have |c τ · x₁ − c τ · x₂| = |c τ| · |x₁ − x₂|
      by (metis abs-mult left-diff-distrib mult.commute)
    also have ... ≤ (max 1 |c t| + |c t|) · |x₁ − x₂|
      using mult-abs-right-mono[OF obs] by blast
    finally have |c τ · x₁ − c τ · x₂| ≤ (max 1 |c t| + |c t|) · |x₁ − x₂| .}
  hence ∃L. ∀t∈cball t (δ/2) ∩ T. 0 ≤ L ∧
    (∀x₁∈cball x (δ/2). ∀x₂∈cball x (δ/2). |c t · x₁ − c t · x₂| ≤ L · |x₁ − x₂|)
    by (rule-tac x=max 1 |c t| + |c t| in exI, clarsimp simp: dist-norm)
  thus ∃u>0. ∃L. ∀t∈cball t u ∩ T. 0 ≤ L ∧
    (∀xa∈cball x u. ∀y∈cball x u. |c t · xa − c t · y| ≤ L · |xa − y|)
    apply(rule-tac x=δ/2 in exI)
    using d-hyp by auto
qed

lemma picard-lindeloef-first-order-linear: t₀ ∈ T ⟹ open T ⟹ is-interval T ⟹
  continuous-on T c ⟹ picard-lindeloef (λt x::real. c t ∗ x) T UNIV t₀
  apply(unfold-locales; clarsimp?)
   apply(intro continuous-intros, assumption)
  by (rule local-lipschitz-first-order-linear)


lemma ⌈λs::realˆ2. s$1 + s$2 = 0⌉ ≤
  wp (x´=(λt s. (χ i. if i=1 then A∗(s$1)ˆ2+B∗(s$1) else A∗(s$2)∗(s$1)+B∗(s$2))) & G on (λs. UNIV)
UNIV @ 0)
  ⌈λs. 0 = − s$1 − s$2⌉
proof−
  have key: diff-invariant (λs. s $ 1 + s $ 2 = 0)
     (λt s. χ i. if i = 1 then A∗(s$1)ˆ2+B∗(s$1) else A∗(s$2)∗(s$1)+B∗(s$2)) (λs. UNIV) UNIV 0 G
  proof(clarsimp simp: diff-invariant-eq ivp-sols-def forall-2)
    fix X::real⇒realˆ2 and t::real
    let ?c = (λt.  X t $ 1 + X t $ 2)
    assume init: ?c 0 = 0
      and D1: D (λt. X t $ 1) = (λt. A · (X t $ 1)² + B · X t $ 1) on UNIV
      and D2: D (λt. X t $ 2) = (λt. A · X t $ 2 · X t $ 1 + B · X t $ 2) on UNIV
    hence D ?c = (λt. ?c t ∗ (A · (X t $ 1) + B)) on UNIV
```

by (*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)
hence *D ?c* = (λt. (*A* · *X t* \$ *1* + *B*) · (*X t* \$ *1* + *X t* \$ *2*)) *on* {*0*−−*t*}
using *has-vderiv-on-subset*[*OF* - *subset-UNIV*[*of* {*0*−−*t*}]] **by** (*simp add*: *mult.commute*)
moreover have *continuous-on UNIV* (λt. *A* · (*X t* \$ *1*) + *B*)
apply(*rule vderiv-on-continuous-on*)
using *D1* **by** (*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)
moreover have *D* (λt. *0*) = (λt. (*A* · *X t* \$ *1* + *B*) · *0*) *on* {*0*−−*t*}
by (*auto intro*!: *poly-derivatives*)
moreover note *picard-lindeloef.ivp-unique-solution*[*OF*
picard-lindeloef-first-order-linear[*OF UNIV-I open-UNIV is-interval-univ calculation*(*2*)]
UNIV-I is-interval-closed-segment-1 subset-UNIV -
ivp-solsI[*OF* - - *funcset-UNIV*, *of ?c*]
ivp-solsI[*OF* - - *funcset-UNIV*, *of* λt. *0*], *of t* λs. *0 0* λs. *t 0*]
ultimately show *X t* \$ *1* + *X t* \$ *2* = *0*
using *init* **by** *auto*
qed
show *?thesis*
apply(*subgoal-tac* (λs. *0* = − *s*\$*1* − *s*\$*2*) = (λs. *s*\$*1* + *s*\$*2* = *0*), *erule ssubst*)
using *key* **by** *auto*
qed

Dynamics: Fractional Darboux equality

lemma *0* ≤ *t* ⟹ ⌈λs::*real*ˆ*3*. *s*\$*1* + *s*\$*3* = *0*⌉ ≤
wp (*x*´= (λt *s*. (χ *i*. *if i*=*1 then* (*A*∗(*s*\$*2*)+*B*∗(*s*\$*1*))/(*s*\$*3*)^2 *else* (*if i* = *3 then* (*A*∗(*s*\$*1*)+*B*)/*s*\$*3 else 0*))) & (λs. (*s*\$*2*) = (*s*\$*1*)ˆ*2* ∧ (*s*\$*3*)ˆ*2* > *0*) *on* (λs. *UNIV*) *UNIV* @ *0*)
⌈λs. *s*\$*1* + *s*\$*3* = *0*⌉
oops

Dynamics: Darboux inequality

lemma ⌈λs::*real*ˆ*3*. *s*\$*1* + *s*\$*3* ≥ *0*⌉ ≤
wp (*x*´= (λt *s*. χ *i*. *if i*=*1 then* (*s*\$*1*)ˆ*2 else* (*if i*=*3 then* (*s*\$*3*)∗(*s*\$*1*)+(*s*\$*2*) *else 0*)) & (λs. *s*\$*2* = (*s*\$*1*)ˆ*2*) *on* (λs. {*0*..}) *UNIV* @ *0*)
⌈λs. *s*\$*1* + *s*\$*3* ≥ *0*⌉
oops

Dynamics: Bifurcation

lemma *picard-lindeloef-dyn-bif*:
continuous-on T (*g*::*real* ⇒ *real*) ⟹ t_0 ∈ *T* ⟹ *is-interval T* ⟹ *open T* ⟹ *picard-lindeloef* (λt τ::*real*. *r* + τˆ*2*) *T UNIV* t_0
proof(*unfold-locales*; *clarsimp simp*: *dist-norm local-lipschitz-def lipschitz-on-def*)
fix *x t*::*real*
{**fix** *x1* **and** *x2*
assume *x1* ∈*cball x 1* **and** *x2* ∈*cball x 1*
hence *leq*: |*x* − *x1*| ≤ *1* |*x* − *x2*| ≤ *1*
by (*auto simp*: *dist-norm*)
have |*x1* + *x2*| = |*x1* − *x* + *x2* − *x* + *2* ∗ *x*|
by *simp*
also have ... ≤ |*x1* − *x*| + |*x2* − *x*| + *2* ∗ |*x*|
using *abs-triangle-ineq* **by** *auto*
also have ... ≤ *2* ∗ (*1* + |*x*|)
using *leq* **by** *auto*
finally have *obs*: |*x1* + *x2*| ≤ *2* ∗ (*1* + |*x*|) **.**
also have |*x1*^{2} − *x2*^{2}| = |*x1* + *x2*|∗|*x1* − *x2*|
by (*metis abs-mult power2-eq-square square-diff-square-factored*)
ultimately have |*x1*^{2} − *x2*^{2}| ≤ (*2* ∗ (*1* + |*x*|)) ∗ |*x1* − *x2*|
by (*metis abs-ge-zero mult-right-mono*)}

thus $\exists\, u{>}0.\ (\exists\,\tau.\ |t - \tau| \leq u \wedge \tau \in T) \longrightarrow (\exists\, L{\geq}0.\ \forall\, xa{\in}cball\ x\ u.\ \forall\, y{\in}cball\ x\ u.\ |xa^2 - y^2| \leq L \cdot |xa - y|)$
by *(rule-tac x=1* **in** *exI, clarsimp, rule-tac x=2 · (1 + |x|)* **in** *exI, auto)*
qed

lemma $r \leq 0 \Longrightarrow \exists\, c.\ \lceil \lambda s{::}real\hat{\ }1.\ s\$1 = c\rceil \leq$
wp (x´= (λt s. (χ i. r + (s$1)ˆ2)) & G on (λs. UNIV) UNIV @ 0)
$\lceil \lambda s.\ s\$1 = c\rceil$
proof*(rule-tac x=sqrt |r|* **in** *exI, clarsimp simp: diff-invariant-eq ivp-sols-def)*
fix *X::real⇒realˆ1* **and** *t::real*
assume *init: X 0 $ 1 = sqrt (− r)* **and** $r \leq 0$
and *D1: D (λx. X x $ 1) = (λx. r + (X x $ 1)²) on UNIV*
hence *D (λx. X x $ 1) = (λx. r + (X x $ 1)²) on {0−−t}*
using *has-vderiv-on-subset* **by** *blast*
moreover have *continuous-on UNIV (λt. X t $ 1)*
apply*(rule vderiv-on-continuous-on)*
using *D1* **by** *assumption*
moreover have *key: D (λt. sqrt (− r)) = (λt. r + (sqrt (− r))²) on {0−−t}*
apply*(subgoal-tac (λt. r + (sqrt (− r))²) = (λt. 0))*
apply*(erule ssubst, rule poly-derivatives)*
using *‹r ≤ 0›* **by** *auto*
moreover note *picard-lindeloef.ivp-unique-solution[OF*
picard-lindeloef-dyn-bif[OF calculation(2) UNIV-I is-interval-univ open-UNIV]
UNIV-I is-interval-closed-segment-1 subset-UNIV -
ivp-solsI[of λx. X x $ 1 - λs. {0−−t} sqrt (− r) 0, OF - - funcset-UNIV]
ivp-solsI[of λt. sqrt (− r) -, OF - - funcset-UNIV], of t r]
ultimately show *X t $ 1 = sqrt (− r)*
using *‹r ≤ 0› init* **by** *auto*
qed

Dynamics: Parametric switching between two different damped oscillators

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

lemma *forall-5*: $(\forall\, i{::}5.\ P\ i) = (P\ 1 \wedge P\ 2 \wedge P\ 3 \wedge P\ 4 \wedge P\ 5)$
by *(metis exhaust-5)*

abbreviation *switch-two-osc-f :: realˆ5 ⇒ realˆ5 (f)*
where *f s ≡ (χ i. if i = 4 then s$5 else (if i = 5 then − (s$3ˆ2) ∗ s$4 − 2 ∗ s$2 ∗ s$3 ∗ s$5 else 0))*

declare *wp-diff-inv [simp del]*

lemma $-\ 2 \leq a \Longrightarrow a \leq 2 \Longrightarrow b\hat{\ }2 \geq 1/3 \Longrightarrow$
$\lceil \lambda s.\ s\$3 \geq 0 \wedge s\$2 \geq 0 \wedge s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1\rceil \leq wp$
(LOOP
((x´= (λt. f) & (λs. True) on (λs. {0..}) UNIV @ 0);
$((\lceil \lambda s.\ s\$4 = s\$5{*}a\rceil;\ (3 ::= (\lambda s.\ 2 * s\$3));\ (2 ::= (\lambda s.\ s\$2/2));\ (1 ::= (\lambda s.\ s\$1 * ((2 \cdot s\$3)^2 + 1)/(s\$3\hat{\ }2 + 1)))) \cup$
$(\lceil \lambda s.\ s\$4 = s\$5{*}b\rceil;\ (3 ::= (\lambda s.\ s\$3/2));\ (2 ::= (\lambda s.\ 2 * s\$2));\ (1 ::= (\lambda s.\ s\$1 * ((s\$3)^2 + 1)/(2 *$

$(s\$3\hat{\ }2) + 1)))) \cup$
$\lceil\lambda s.\ True\rceil))$
$INV\ (\lambda s.\ s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0))$
$\lceil\lambda s.\ s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1\rceil$
apply(*subst change-loopI*[**where** $I{=}\lambda s.\ s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$])
apply(*rule wp-loopI*, *simp-all add*: *le-wp-choice-iff*, *intro conjI*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$])
apply(*rule wp-g-odei*, *simp*)
apply(*rule-tac* $C{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$], *rule wp-g-odei*; *simp add*: *wp-diff-inv*)
apply(*rule diff-invariant-conj-rule*)
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
apply(*simp add*: *wp-diff-inv*, *intro diff-invariant-conj-rule*)
apply(*rule-tac* $\nu'{=}\lambda s.\ -4*(s\$2)*(s\$3)*(s\$5)\hat{\ }2$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2); (*clarsimp simp*: *forall-5*)?)
apply(*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)[1]
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
subgoal sorry
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$3\hat{\ }2 * s\$4\hat{\ }2 + s\$5\hat{\ }2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$])
apply(*rule wp-g-odei*, *simp*)
apply(*rule-tac* $C{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$], *rule wp-g-odei*, *simp-all*)
apply(*simp add*: *wp-diff-inv*, *rule diff-invariant-conj-rule*)
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
apply(*simp add*: *wp-diff-inv*, *intro diff-invariant-conj-rule*)
apply(*rule-tac* $\nu'{=}\lambda s.\ -4*(s\$2)*(s\$3)*(s\$5)\hat{\ }2$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2); (*clarsimp simp*: *forall-5*)?)
apply(*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)[1]
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
subgoal sorry
apply(*rule-tac* $C{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$ **in** *diff-cut-rule*, *simp-all*)
apply(*subst g-ode-inv-def*[*symmetric*, **where** $I{=}\lambda s.\ s\$2 \geq 0 \wedge s\$3 \geq 0$], *rule wp-g-odei*, *simp-all*)
apply(*simp add*: *wp-diff-inv*, *rule diff-invariant-conj-rule*)
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
apply(*simp add*: *wp-diff-inv*, *intro diff-invariant-conj-rule*)
apply(*rule-tac* $\nu'{=}\lambda s.\ -4*(s\$2)*(s\$3)*(s\$5)\hat{\ }2$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2); (*clarsimp simp*: *forall-5*)?)
apply(*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)[1]
apply (*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 0$ **in** *diff-invariant-rules*(2), *simp-all add*: *forall-5*)+
done

declare *wp-diff-inv* [*simp*]

no-notation *switch-two-osc-f* (*f*)

Dynamics: Nonlinear 1

lemma $\lceil\lambda s{::}real\hat{\ }1.\ s\$1\hat{\ }3 \geq -1\rceil \leq wp$
$(x\acute{} = (\lambda s.\ \chi\ i.\ (s\$1 - 3)\hat{\ }4 + a)\ \&\ (\lambda s.\ a \geq 0))$
$\lceil\lambda s.\ s\$1\hat{\ }3 \geq -1\rceil$
apply(*simp*, *rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 3 * s\$1\hat{\ }2 * ((s\$1 - 3)\hat{\ }4 + a)$ **in** *diff-invariant-rules*(2))
by (*auto intro*!: *poly-derivatives simp*: *field-simps*)

Dynamics: Nonlinear 2

lemma $\lceil\lambda s{::}real\hat{\ }2.\ s\$1 + (s\$2\hat{\ }2)/2 = a\rceil \leq$
$wp\ (x\acute{} = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$1 * s\$2\ else - s\$1)\ \&\ G)$

$\lceil\lambda s.\ s\$1 + (s\$2\hat{}2)/2 = a\rceil$
by (*auto intro*!: *diff-invariant-rules poly-derivatives*)

Dynamics: Nonlinear 4

lemma $\lceil\lambda s{::}real\hat{}2.\ (s\$1)\hat{}2/2 - (s\$2\hat{}2)/2 \geq a\rceil \leq$
wp $(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2 + s\$1 * (s\$2\hat{}2)\ else - s\$1 + s\$1\hat{}2 * s\$2)\ \&\ (\lambda s.\ s\$1 \geq 0 \wedge s\$2 \geq 0))$
$\lceil\lambda s.\ (s\$1)\hat{}2/2 - (s\$2\hat{}2)/2 \geq a\rceil$
apply(*simp*, *rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ s\$1{*}(s\$2 + s\$1 * (s\$2\hat{}2)) - s\$2 * (- s\$1 + s\$1\hat{}2 * s\$2)$ **in** *diff-invariant-rules*(2))
by (*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)

Dynamics: Nonlinear 5

lemma $\lceil\lambda s{::}real\hat{}2.\ -(s\$1) * (s\$2) \geq a\rceil \leq$
wp $(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$1 - s\$2 + s\$1 * s\$2\ else - s\$2 - s\$2\hat{}2)\ \&\ G)$
$\lceil\lambda s.\ -(s\$1){*}(s\$2) \geq a\rceil$
apply(*simp*, *rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ (- s\$1 + s\$2 - s\$1 * s\$2) * s\$2 - s\$1 * (- s\$2 - s\$2\hat{}2)$ **in** *diff-invariant-rules*(2))
by (*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)

Dynamics: Riccati

lemma $\lceil\lambda s{::}real\hat{}1.\ 2 * s\$1\hat{}3 \geq 1/4\rceil \leq$
wp $(x' = (\lambda s.\ \chi\ i.\ s\$1\hat{}2 + s\$1\hat{}4)\ \&\ G)$
$\lceil\lambda s.\ 2 * s\$1\hat{}3 \geq 1/4\rceil$
apply(*simp*, *rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 24 * (s\$1\hat{}2) * (s\$1\hat{}2 + s\$1\hat{}4)$ **in** *diff-invariant-rules*(2); *clarsimp*)
by (*auto intro*!: *poly-derivatives simp*: *field-simps power2-eq-square*)

Dynamics: Nonlinear differential cut

lemma $\lceil\lambda s{::}real\hat{}2.\ s\$1\hat{}3 \geq - 1 \wedge s\$2\hat{}5 \geq 0\rceil \leq$
wp $(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ (s\$1 - 3)\hat{}4\ else\ s\$2\hat{}2)\ \&\ G)$
$\lceil\lambda s.\ s\$1\hat{}3 \geq - 1 \wedge s\$2\hat{}5 \geq 0\rceil$
apply(*simp*, *rule diff-invariant-rules*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ 3 * s\$1\hat{}2 * (s\$1 - 3)\hat{}4$ **in** *diff-invariant-rules*(2))
apply(*simp-all add*: *forall-2*, *force intro*!: *poly-derivatives*)
apply(*rule-tac* $\nu'{=}\lambda s.\ 0$ **and** $\mu'{=}\lambda s.\ s\$2\hat{}2$ **in** *diff-invariant-rules*(2))
by (*auto intro*!: *diff-invariant-rules poly-derivatives simp*: *forall-2*)

STTT Tutorial: Example 1

lemma $A > 0 \Longrightarrow \lceil\lambda s{::}real\hat{}2.\ s\$2 \geq 0\rceil \leq$
wp $(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ A)\ \&\ G)$
$\lceil\lambda s.\ s\$2 \geq 0\rceil$
apply(*subst local-flow.wp-g-ode-subset*[**where** $T{=}UNIV$
and $\varphi{=}\lambda t\ s.\ \chi\ i{::}2.\ if\ i{=}1\ then\ A * t\hat{}2/2 + s\$2 * t + s\$1\ else\ A * t + s\$2$])
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def forall-2 lipschitz-on-def*)
apply(*clarsimp*, *rule-tac* $x{=}1$ **in** *exI*)+
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*)
unfolding *UNIV-2* **using** *exhaust-2* **by** (*auto intro*!: *poly-derivatives simp*: *vec-eq-iff*)

STTT Tutorial: Example 2

lemma *local-flow-STTT-Ex2*:
local-flow $(\lambda s{::}real\hat{}3.\ \chi\ i.\ if\ i = 1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ 0))$ *UNIV UNIV*
$(\lambda t\ s.\ \chi\ i.\ if\ i = 1\ then\ s\$3 * t\hat{}2/2 + s\$2 * t + s\$1\ else\ (if\ i{=}2\ then\ s\$3 * t + s\$2\ else\ s\$i))$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def forall-2 lipschitz-on-def*)
apply(*clarsimp*, *rule-tac* $x{=}1$ **in** *exI*)+

apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*)
unfolding *UNIV-3* **by** (*auto intro*!: *poly-derivatives simp*: *forall-3 vec-eq-iff*)

lemma $A > 0 \Longrightarrow B > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }3.\ s\$2 \geq 0\rceil \leq wp$
$(LOOP\ ($
$(((3 ::= (\lambda s.\ A)) \cup (3 ::= (\lambda s.\ 0)) \cup (3 ::= (\lambda s.\ B)));$
$(x' = (\lambda s.\ \chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ s\$3\ \mathit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0)))$
$)\ INV\ (\lambda s.\ s\$2 \geq 0))$
$\lceil \lambda s.\ s\$2 \geq 0\rceil$
apply(*rule wp-loopI*, *simp-all add*: *le-wp-choice-iff*, *intro conjI*)
by (*simp-all add*: *local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])

STTT Tutorial: Example 3a

lemma *STTexample3a-arith*:
assumes $0 < (B{::}real)$ $0 \leq t$ $0 \leq x2$ **and** *key*: $x1 + x2^2 / (2 \cdot B) \leq S$
shows $x2 \cdot t - B \cdot t^2 / 2 + x1 + (x2 - B \cdot t)^2 / (2 \cdot B) \leq S$ (**is** $?lhs \leq S$)
proof−
have $?lhs = 2 * B * x2 \cdot t/(2{*}B) - B\hat{\ }2 \cdot t^2 / (2{*}B) + (2 * B * x1)/(2{*}B) + (x2 - B \cdot t)^2 / (2 \cdot B)$
using ‹$0 < B$› **by** (*auto simp*: *power2-eq-square*)
also have $(x2 - B \cdot t)^2 / (2 \cdot B) = x2\hat{\ }2/(2{*}B) + B\hat{\ }2 * t\hat{\ }2/(2{*}B) - 2{*}x2{*}B{*}t/(2{*}B)$
using ‹$0 < B$› **by** (*auto simp*: *power2-diff field-simps*)
ultimately have $?lhs = x1 + x2^2 / (2 \cdot B)$
using ‹$0 < B$› **by** *auto*
thus $?lhs \leq S$
using *key* **by** *simp*
qed

lemma $A > 0 \Longrightarrow B > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }3.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{\ }2/(2{*}B) < S\rceil \leq wp$
$(LOOP\ ($
$((\lceil \lambda s.\ s\$1 + s\$2\hat{\ }2/(2{*}B) < S\rceil;(3 ::= (\lambda s.\ A))) \cup (\lceil \lambda s.\ s\$2 = 0\rceil;(3 ::= (\lambda s.\ 0))) \cup (3 ::= (\lambda s.\ -B)));$
$((x' = (\lambda s.\ \chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ s\$3\ \mathit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{\ }2/(2{*}B) \leq S)) \cup$
$(x' = (\lambda s.\ \chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ s\$3\ \mathit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{\ }2/(2{*}B) \geq S)))$
$)\ INV\ (\lambda s.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{\ }2/(2{*}B) \leq S))$
$\lceil \lambda s.\ s\$1 \leq S\rceil$
apply(*rule wp-loopI*)
apply(*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])
apply *safe*
apply (*smt not-sum-power2-lt-zero zero-compare-simps*(*5*))
apply(*erule-tac* $x{=}0$ **in** *allE*)
by (*auto simp*: *STTexample3a-arith*)

STTT Tutorial: Example 4a

lemma $A > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }3.\ s\$2 \leq V\rceil \leq wp$
$(LOOP$
$(\lceil \lambda s.\ s\$2 = V\rceil;(3 ::= (\lambda s.\ 0))) \cup (\lceil \lambda s.\ s\$2 \neq V\rceil;(3 ::= (\lambda s.\ A)));$
$(x' = (\lambda s.\ \chi\ i.\ \mathit{if}\ i{=}1\ \mathit{then}\ s\$2\ \mathit{else}\ (\mathit{if}\ i{=}2\ \mathit{then}\ s\$3\ \mathit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \leq V))$
$INV\ (\lambda s.\ s\$2 \leq V))$
$\lceil \lambda s.\ s\$2 \leq V\rceil$
by (*rule wp-loopI*)
(*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])

STTT Tutorial: Example 4b

lemma $A > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }3.\ s\$2 \leq V\rceil \leq wp$
$(LOOP$

$(3 ::= (\lambda s.\ A))$;
$(x' = (\lambda s.\ \chi\ i.\ \textit{if}\ i{=}1\ \textit{then}\ s\$2\ \textit{else}\ (\textit{if}\ i{=}2\ \textit{then}\ s\$3\ \textit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \leq V))$
INV $(\lambda s.\ s\$2 \leq V))$
$\lceil \lambda s.\ s\$2 \leq V \rceil$
by (*rule wp-loopI*) (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])

STTT Tutorial: Example 4c

lemma $A > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }3.\ s\$2 \leq V \rceil \leq wp$
(*LOOP*
$(\lceil \lambda s.\ s\$2 = V \rceil;(3 ::= (\lambda s.\ 0))) \cup (\lceil \lambda s.\ s\$2 \neq V \rceil;(3 ::= (\lambda s.\ A)))$;
$((x' = (\lambda s.\ \chi\ i.\ \textit{if}\ i{=}1\ \textit{then}\ s\$2\ \textit{else}\ (\textit{if}\ i{=}2\ \textit{then}\ s\$3\ \textit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \leq V)) \cup$
$(x' = (\lambda s.\ \chi\ i.\ \textit{if}\ i{=}1\ \textit{then}\ s\$2\ \textit{else}\ (\textit{if}\ i{=}2\ \textit{then}\ s\$3\ \textit{else}\ 0))\ \&\ (\lambda s.\ s\$2 \geq V)))$
INV $(\lambda s.\ s\$2 \leq V))$
$\lceil \lambda s.\ s\$2 \leq V \rceil$
apply (*rule wp-loopI*)
apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])
by (*clarsimp*, *erule-tac* $x{=}0$ **in** *allE*, *auto*)

STTT Tutorial: Example 5

lemma *STTexample5-arith*:
assumes $0 < A$ $0 < B$ $0 < \varepsilon$ $0 \leq x2$ $0 \leq (t{::}real)$
and *key*: $x1 + x2^2 / (2 \cdot B) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2) / B + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2)) \leq S$ (**is** $?k3 \leq S$)
and *ghyp*: $\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon$
shows $A \cdot t^2 / 2 + x2 \cdot t + x1 + (A \cdot t + x2)^2 / (2 \cdot B) \leq S$ (**is** $?k0 \leq S$)
proof−
have $t \leq \varepsilon$
using *ghyp* $\langle 0 \leq t \rangle$ **by** *auto*
hence $A{*}t\hat{\ }2/2 + t{*}x2 \leq A{*}\varepsilon\hat{\ }2/2 + \varepsilon{*}x2$
using $\langle 0 \leq t \rangle$ $\langle 0 < A \rangle$ $\langle 0 \leq x2 \rangle$
by (*smt field-sum-of-halves mult-right-mono power-less-imp-less-base real-mult-le-cancel-iff2*)
hence $((A + B)/B) * (A{*}t\hat{\ }2/2 + t{*}x2) + x1 + x2^2 / (2 \cdot B) \leq$
$x1 + x2^2 / (2 \cdot B) + ((A + B)/B) * (A{*}\varepsilon\hat{\ }2/2 + \varepsilon{*}x2)$ (**is** $?k1 \leq ?k2$)
using $\langle 0 < B \rangle$ $\langle 0 < A \rangle$ **by** (*smt real-mult-le-cancel-iff2 zero-compare-simps*(*9*))
moreover have $?k0 = ?k1$
using $\langle 0 < B \rangle$ $\langle 0 < A \rangle$ **by** (*auto simp*: *field-simps power2-sum power2-eq-square*)
moreover have $?k2 = ?k3$
using $\langle 0 < B \rangle$ $\langle 0 < A \rangle$ **by** (*auto simp*: *field-simps power2-sum power2-eq-square*)
ultimately show $?k0 \leq S$
using *key* **by** *linarith*
qed

lemma *local-flow-STTT-Ex5*:
local-flow $(\lambda s{::}real\hat{\ }4.\ \chi\ i.\ \textit{if}\ i{=}1\ \textit{then}\ s\$2\ \textit{else}\ (\textit{if}\ i{=}2\ \textit{then}\ s\$3\ \textit{else}\ (\textit{if}\ i{=}3\ \textit{then}\ 0\ \textit{else}\ 1)))$ *UNIV UNIV*
$(\lambda t\ s.\ \chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ s\$3 * t\hat{\ }2/2 + s\$2 * t + s\$1\ \textit{else}\ (\textit{if}\ i{=}2\ \textit{then}\ s\$3 * t + s\$2\ \textit{else}\ (\textit{if}\ i{=}3\ \textit{then}\ s\$3\ \textit{else}\ t{+}s\$4)))$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def forall-2 lipschitz-on-def*)
apply(*clarsimp*, *rule-tac* $x{=}1$ **in** *exI*)+
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*)
unfolding *UNIV-4* **by** (*auto intro*!: *poly-derivatives simp*: *forall-4 vec-eq-iff*)

lemma $A > 0 \Longrightarrow B > 0 \Longrightarrow \varepsilon > 0 \Longrightarrow \lceil \lambda s{::}real\hat{\ }4.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{\ }2/(2{*}B) \leq S \rceil \leq wp$
(*LOOP*
(
$(\lceil \lambda s.\ s\$1 + s\$2\hat{\ }2/(2{*}B) + (A/B + 1) * (A/2 * \varepsilon\hat{\ }2 + \varepsilon * s\$2) \leq S \rceil;(3 ::= (\lambda s.\ A))) \cup$
$(\lceil \lambda s.\ s\$2 = 0 \rceil;(3 ::= (\lambda s.\ 0))) \cup$
$(3 ::= (\lambda s.\ -B)))$;

(*4* ::= (λ*s*. *0*));
(*x´*= (λ*s*. χ *i*. *if i=1 then s$2 else* (*if i=2 then s$3 else* (*if i=3 then 0 else 1*))) & (λ*s*. *s$2* ≥ *0* ∧ *s$4* ≤ ε))
INV (λ*s*. *s$2* ≥ *0* ∧ *s$1* + *s$2*ˆ*2*/(*2*∗*B*) ≤ *S*))
⌈λ*s*. *s$1* ≤ *S*⌉
apply (*rule wp-loopI*)
apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*])
apply *safe*
apply (*smt not-sum-power2-lt-zero zero-compare-simps*(*5*))
by (*auto simp*: *STTexample3a-arith STTexample5-arith*)

STTT Tutorial: Example 6

lemma *STTexample6-arith*:
assumes $0 < A$ $0 < B$ $0 < \varepsilon$ $0 \le x2$ $0 \le (t{::}real) - B \le k$ $k \le A$
and *key*: $x1 + x2^2 / (2 \cdot B) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2) / B + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2)) \le S$ (**is** *?k3* ≤ *S*)
and *ghyp*: $\forall \tau.\ 0 \le \tau \wedge \tau \le t \longrightarrow 0 \le k \cdot \tau + x2 \wedge \tau \le \varepsilon$
shows $k \cdot t^2 / 2 + x2 \cdot t + x1 + (k \cdot t + x2)^2 / (2 \cdot B) \le S$ (**is** *?k0* ≤ *S*)
proof−
have $0 \le k \cdot t + x2 + x2$
using *ghyp* ‹*0* ≤ *x2*› ‹*0* ≤ *t*› **by** *force*
hence $0 \le (k \cdot t + 2 * x2) * t/2$
by (*metis assms*(*5*) *divide-nonneg-pos is-num-normalize*(*1*) *mult-2 mult-sign-intros*(*1*) *rel-simps*(*51*))
hence *f1*: $0 \le k{*}t\hat{}2/2 + t{*}x2$
by (*auto simp*: *field-simps power2-eq-square*)
have *f2*: $0 \le (k + B) / B$ $(k + B) / B \le (A + B) / B$
using ‹*0* < *A*› ‹*0* < *B*› ‹− *B* ≤ *k*› ‹*k* ≤ *A*› *divide-le-cancel* **by** (*auto*, *fastforce*)
have $t \le \varepsilon$
using *ghyp* ‹*0* ≤ *t*› **by** *auto*
hence $k{*}t\hat{}2/2 + t{*}x2 \le A{*}t\hat{}2/2 + t{*}x2$
using ‹*k* ≤ *A*› **by** (*auto simp*: *mult-right-mono*)
also have *f3*: ... $\le A{*}\varepsilon\hat{}2/2 + \varepsilon{*}x2$
using ‹*0* ≤ *t*› ‹*0* < *A*› ‹*0* ≤ *x2*› ‹*t* ≤ ε›
by (*smt field-sum-of-halves mult-right-mono power-less-imp-less-base real-mult-le-cancel-iff2*)
finally have $k{*}t\hat{}2/2 + t{*}x2 \le A{*}\varepsilon\hat{}2/2 + \varepsilon{*}x2$ **.**
hence $((k + B)/B) * (k{*}t\hat{}2/2 + t{*}x2) \le ((A + B)/B) * (A{*}\varepsilon\hat{}2/2 + \varepsilon{*}x2)$
using *f1 f2* ‹*k* ≤ *A*› **apply**(*rule-tac b*=((*A* + *B*)/*B*) ∗ (*A*∗*t*ˆ*2*/*2* + *t*∗*x2*) **in** *order.trans*)
apply (*rule mult-mono′*, *simp*, *simp*, *simp add*: *mult-right-mono*, *simp*, *simp*)
by (*metis f3 add-sign-intros*(*4*) *assms*(*1*,*2*) *less-eq-real-def mult-zero-left*
real-mult-le-cancel-iff2 zero-compare-simps(*5*))
hence $((k + B)/B) * (k{*}t\hat{}2/2 + t{*}x2) + x1 + x2^2 / (2 \cdot B) \le$
$x1 + x2^2 / (2 \cdot B) + ((A + B)/B) * (A{*}\varepsilon\hat{}2/2 + \varepsilon{*}x2)$ (**is** *?k1* ≤ *?k2*)
using ‹*0* < *B*› ‹*0* < *A*› **by** (*smt real-mult-le-cancel-iff2 zero-compare-simps*(*9*))
moreover have *?k0* = *?k1*
using ‹*0* < *B*› ‹*0* < *A*› **by** (*auto simp*: *field-simps power2-sum power2-eq-square*)
moreover have *?k2* = *?k3*
using ‹*0* < *B*› ‹*0* < *A*› **by** (*auto simp*: *field-simps power2-sum power2-eq-square*)
ultimately show *?k0* ≤ *S*
using *key* **by** *linarith*
qed

lemma $A > 0 \Longrightarrow B > 0 \Longrightarrow \varepsilon > 0 \Longrightarrow$ ⌈λ*s*::*real*ˆ*4*. *s$2* ≥ *0* ∧ *s$1* + *s$2*ˆ*2*/(*2*∗*B*) ≤ *S*⌉ ≤ *wp*
(*LOOP*
(
(⌈λ*s*. *s$1* + *s$2*ˆ*2*/(*2*∗*B*) + (*A*/*B* + *1*) ∗ (*A*/*2* ∗ εˆ*2* + ε ∗ *s$2*) ≤ *S*⌉;(*3* ::= *?*);⌈λ*s*. −*B* ≤ *s$3* ∧ *s$3* ≤ *A*⌉) ∪
(⌈λ*s*. *s$2* = *0*⌉;(*3* ::= (λ*s*. *0*))) ∪
(*3* ::= (λ*s*. − *B*)));

$(4 ::= (\lambda s.\ 0));$
$(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ (if\ i{=}3\ then\ 0\ else\ 1)))\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$
$INV\ (\lambda s.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{}2/(2{*}B) \leq S))$
$\lceil\lambda s.\ s\$1 \leq S\rceil$
apply (*rule wp-loopI*)
apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*])
apply *safe*
apply (*smt not-sum-power2-lt-zero zero-compare-simps*(*5*))
by (*auto simp*: *STTexample3a-arith STTexample6-arith*)

STTT Tutorial: Example 7

lemma *STTexample7-arith1*:
assumes $(0{::}real) < A$ $0 < b$ $0 < \varepsilon$ $0 \leq v$ $0 \leq t$ $k \leq A$
and $x + v^2 / (2 \cdot b) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) / b + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v)) \leq S$ (**is** $?expr1 \leq S$)
and *guard*: $\forall\tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq k \cdot \tau + v \wedge \tau \leq \varepsilon$
shows $k \cdot t^2 / 2 + v \cdot t + x + (k \cdot t + v)^2 / (2 \cdot b) \leq S$ (**is** $?lhs \leq S$)
proof−
have *obs1*: $?lhs \cdot (2 \cdot b) = k \cdot t^2 \cdot b + 2 \cdot v \cdot t \cdot b + 2 \cdot x \cdot b + (k \cdot t + v)^2$ (**is** - $= ?expr2\ k\ t$)
using ‹$0 < b$› **by** (*simp add*: *field-simps*)
have $?expr2\ A\ \varepsilon = ?expr1 \cdot (2 \cdot b)$
using ‹$0 < b$› **by** (*simp add*: *field-simps power2-eq-square*)
also have ... $\leq S \cdot (2 \cdot b)$
using ‹$?expr1 \leq S$› ‹$0 < b$› **by** (*smt real-mult-less-iff1*)
finally have *obs2*: $?expr2\ A\ \varepsilon \leq S \cdot (2 \cdot b)$ **.**
have $t \leq \varepsilon$
using *guard* ‹$0 \leq t$› **by** *auto*
hence $t^2 \leq \varepsilon^2$ $k \cdot t + v \leq A \cdot \varepsilon + v$
using ‹$k \leq A$› ‹$0 < A$› *power-mono*[*OF* ‹$t \leq \varepsilon$› ‹$0 \leq t$›, *of 2*]
by *auto* (*meson* ‹$0 \leq t$› *less-eq-real-def mult-mono*)
hence $k \cdot t^2 \cdot b \leq A \cdot \varepsilon^2 \cdot b$ $2 \cdot v \cdot t \cdot b \leq 2 \cdot v \cdot \varepsilon \cdot b$
using ‹$t \leq \varepsilon$› ‹$0 < b$› ‹$k \leq A$› ‹$0 \leq v$›
by (*auto simp*: *mult-left-mono*) (*meson* ‹$0 < A$› *less-eq-real-def mult-mono zero-compare-simps*(*12*))
hence $?expr2\ k\ t \leq ?expr2\ A\ \varepsilon$
by (*smt* ‹$k \cdot t + v \leq A \cdot \varepsilon + v$› *ends-in-segment*(*2*) ‹$0 \leq t$› *guard power-mono*)
hence $?lhs \cdot (2 \cdot b) \leq S \cdot (2 \cdot b)$
using *obs1 obs2* **by** *simp*
thus $?lhs \leq S$
using ‹$0 < b$› **by** (*smt real-mult-less-iff1*)
qed

lemma *STTexample7-arith2*:
assumes $(0{::}real) < b$ $0 \leq v$ $0 \leq t$ $k \leq -b$
and *key*: $x + v^2 / (2 \cdot b) \leq S$
and *guard*: $\forall\tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq k \cdot \tau + v \wedge \tau \leq \varepsilon$
shows $k \cdot t^2 / 2 + v \cdot t + x + (k \cdot t + v)^2 / (2 \cdot b) \leq S$ (**is** $?lhs \leq S$)
proof−
have *obs*: $1 + k/b \leq 0$ $k \cdot t + v \geq 0$
using ‹$k \leq -b$› ‹$0 < b$› *guard* ‹$0 \leq t$› **by** (*auto simp*: *mult-imp-div-pos-le real-add-le-0-iff*)
have $?lhs = (k \cdot t + v + v) \cdot t/2 \cdot (1 + k/b) + x + v^2 / (2 \cdot b)$
using ‹$0 < b$› **by** (*auto simp*: *field-simps power2-eq-square*)
also have ... $\leq x + v^2 / (2 \cdot b)$
using *obs* ‹$0 \leq t$› ‹$0 \leq v$›
by (*smt mult-nonneg-nonneg zero-compare-simps*(*11*) *zero-compare-simps*(*6*))
also have ... $\leq S$
using *key* **.**
finally show *?thesis* **.**
qed

lemma $A > 0 \Longrightarrow B \geq b \Longrightarrow b > 0 \Longrightarrow \varepsilon > 0 \Longrightarrow \lceil\lambda s{::}real\hat{~}4.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{~}2/(2*b) \leq S\rceil \leq$ *wp*
(*LOOP*
(
$(\lceil\lambda s.\ s\$1 + s\$2\hat{~}2/(2*b) + (A/b + 1) * (A/2 * \varepsilon\hat{~}2 + \varepsilon * s\$2) \leq S\rceil;(3 ::= ?);\lceil\lambda s.\ -B \leq s\$3 \wedge s\$3 \leq A\rceil) \cup$
$(\lceil\lambda s.\ s\$2 = 0\rceil;(3 ::= (\lambda s.\ 0))) \cup$
$((3 ::= ?);\lceil\lambda s.\ -B \leq s\$3 \wedge s\$3 \leq -\ b\rceil)$
$);(4 ::= (\lambda s.\ 0));$
$(x' = (\lambda s.\ \chi\ i.$ *if* $i=1$ *then* $s\$2$ *else* (*if* $i=2$ *then* $s\$3$ *else* (*if* $i=3$ *then* 0 *else* 1))) & $(\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$
INV $(\lambda s.\ s\$2 \geq 0 \wedge s\$1 + s\$2\hat{~}2/(2*b) \leq S))$
$\lceil\lambda s.\ s\$1 \leq S\rceil$
apply (*rule wp-loopI*)
apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*])
apply(*safe*)
apply (*smt not-sum-power2-lt-zero zero-compare-simps*(*5*))
apply(*force simp*: *STTexample7-arith1*, *force*)
using *STTexample7-arith2*[*of b -\$2 - - -\$1 S*] **by** *blast*

STTT Tutorial: Example 9a

lemma *STTexample9a-arith*:
$(10{\cdot}x{-}10{\cdot}r){\cdot}v/4 + v^2/2 + (x{-}r){\cdot}(2{\cdot}r{-}2{\cdot}x{-}3{\cdot}v)/2 + v{\cdot}(2{\cdot}r{-}2{\cdot}x{-}3{\cdot}v)/2 \leq (0{::}real)$ (**is** $?t1 + ?t2 + ?t3 + ?t4 \leq 0$)
proof−
have $?t1 = 5 * (x{-}r) * v/2$
by *auto*
moreover have $?t3 = -((x - r)\hat{~}2) - 3 * v * (x{-}r)/2$
by (*auto simp*: *field-simps power2-diff power2-eq-square*)
moreover have $?t4 = -\ 2 * (x - r) * v/2 - 3 * v\hat{~}2/2$
by (*auto simp*: *field-simps power2-diff power2-eq-square*)
ultimately have $?t1 + ?t3 + ?t4 = -((x - r)\hat{~}2) - 3 * v\hat{~}2/2$
by (*auto simp*: *field-simps*)
hence $?t1 + ?t2 + ?t3 + ?t4 = -((x - r)\hat{~}2) - v\hat{~}2$
by *auto*
also have $... \leq 0$
by *auto*
finally show *?thesis* **.**
qed

lemma $c > 0 \Longrightarrow Kp = 2 \Longrightarrow Kd = 3 \Longrightarrow \lceil\lambda s{::}real\hat{~}2.\ (5/4)*(s\$1{-}xr)\hat{~}2 + (s\$1{-}xr)*(s\$2)/2 + (s\$2)\hat{~}2/4 < c\rceil \leq$ *wp*
$(x' = (\lambda s.\ \chi\ i.$ *if* $i=1$ *then* $s\$2$ *else* $-Kp*(s\$1{-}xr) - Kd*(s\$2)\ )\ \&\ G)$
$\lceil\lambda s.\ (5/4)*(s\$1{-}xr)\hat{~}2 + (s\$1{-}xr)*(s\$2)/2 + (s\$2)\hat{~}2/4 < c\rceil$
apply(*simp*, *rule-tac* $\mu'{=}\lambda s.\ 0$ **and** $\nu'{=}\lambda s.\ 10*(s\$1{-}xr)*(s\$2)/4 + (s\$2\hat{~}2)/2 +$
$(s\$1{-}xr)*(-Kp*(s\$1{-}xr){-}Kd*(s\$2))/2 + (s\$2)*(-Kp*(s\$1{-}xr){-}Kd*(s\$2))/2$ **in** *diff-invariant-rules*(*3*);

clarsimp simp: *forall-2 STTexample9a-arith*)
apply(*intro poly-derivatives*; (*rule poly-derivatives*)*?*)
by *force+* (*auto simp*: *field-simps power2-eq-square*)

STTT Tutorial: Example 9b

lemma $c > 0 \Longrightarrow Kp = 2 \Longrightarrow Kd = 3 \Longrightarrow$
$\lceil\lambda s{::}real\hat{~}4.\ s\$2 \geq 0 \wedge s\$3 \leq s\$1 \wedge s\$1 \leq S \wedge s\$4 = (s\$3 + S)/2$
$\wedge (5/4)*(s\$1{-}s\$4)\hat{~}2 + (s\$1{-}s\$4)*(s\$2)/2 + (s\$2)\hat{~}2/4 < ((S - s\$3)/2)\hat{~}2\rceil \leq$ *wp*
(*LOOP* $((3 ::= (\lambda s.\ s\$1));(4 ::= (\lambda s.\ (s\$3 + S)/2));$

$\lceil\lambda s.\ (5/4)*(s\$1-s\$4)\hat{\ }2 + (s\$1-s\$4)*(s\$2)/2 + (s\$2)\hat{\ }2/4 < ((S - s\$3)/2)\hat{\ }2\rceil \cup \lceil\lambda s.\ True\rceil);$
$(x\acute{\ }= (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ -Kp*(s\$1-s\$3) - Kd*(s\$2)\ else\ 0))\ \&$
$(\lambda s.\ s\$2 \geq 0)\ DINV\ (\lambda s.\ s\$3 \leq s\$1 \wedge$
$(5/4)*(s\$1-(s\$3+S)/2)\hat{\ }2 + (s\$1-(s\$3+S)/2)*(s\$2)/2 + s\$2\hat{\ }2/4 < ((S - s\$3)/2)\hat{\ }2))$
$INV\ (\lambda s.\ s\$2 \geq 0 \wedge s\$3 \leq s\$1 \wedge s\$4 = (s\$3 + S)/2 \wedge$
$(5/4)*(s\$1-s\$4)\hat{\ }2 + (s\$1-s\$4)*(s\$2)/2 + (s\$2)\hat{\ }2/4 < ((S - s\$3)/2)\hat{\ }2))$
$\lceil\lambda s.\ s\$1 \leq S\rceil$
oops

STTT Tutorial: Example 10

LICS: Example 1 Continuous car accelerates forward

lemma $\lceil\lambda s{::}real\hat{\ }3.\ s\$2 \geq 0 \wedge s\$3 \geq 0\rceil \leq wp$
$(x\acute{\ }= (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0))$
$\lceil\lambda s.\ s\$2 \geq 0\rceil$
by (*simp-all add*: *local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*])

LICS: Example 2 Single car drives forward

lemma $A \geq 0 \Longrightarrow b > 0 \Longrightarrow \lceil\lambda s{::}real\hat{\ }3.\ s\$2 \geq 0\rceil \leq wp$
$(LOOP$
$((3 ::= (\lambda s.\ A)) \cup (3 ::= (\lambda s.\ -b)));$
$(x\acute{\ }= (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0))$
$INV\ (\lambda s.\ s\$2 \geq 0))$
$\lceil\lambda s.\ s\$2 \geq 0\rceil$
by (*rule wp-loopI*) (*simp-all add*: *local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*] *rel-aka.fbox-add2*)

LICS: Example 3a event-triggered car drives forward

lemma $A \geq 0 \Longrightarrow b > 0 \Longrightarrow \lceil\lambda s{::}real\hat{\ }3.\ s\$2 \geq 0\rceil \leq wp$
$(LOOP$
$((\lceil\lambda s.\ m - s\$1 \geq 2\rceil;(3 ::= (\lambda s.\ A))) \cup (3 ::= (\lambda s.\ -b)));$
$(x\acute{\ }= (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ 0))\ \&\ (\lambda s.\ s\$2 \geq 0))$
$INV\ (\lambda s.\ s\$2 \geq 0))$
$\lceil\lambda s.\ s\$2 \geq 0\rceil$
by (*rule wp-loopI*) (*simp-all add*: *local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex2*] *rel-aka.fbox-add2*)

LICS: Example 4a safe stopping of time-triggered car

lemma *LICSexample4a-arith*:
assumes $(0{::}real) \leq A\ \ 0 < b\ \ v^2 \leq 2 \cdot b \cdot (m - x)\ \ 0 \leq t$
and *guard*: $\forall\tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq A \cdot \tau + v \wedge \tau \leq \varepsilon$
and *key*: $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$ (**is** *?expr1* $\leq$ -)
shows $(A \cdot t + v)^2 \leq 2 \cdot b \cdot (m - (A \cdot t^2 / 2 + v \cdot t + x))$
proof−
have $t \leq \varepsilon\ \ 0 \leq v$
using *guard* ‹$0 \leq t$› **by** (*force*, *erule-tac x=0* **in** *allE*, *auto*)
hence $A \cdot t^2 + 2 \cdot t \cdot v \leq A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v$
using ‹$0 \leq A$› ‹$0 \leq t$›
by (*smt mult-less-cancel-left-disj mult-right-mono power-less-imp-less-base*)
hence $v^2 + (A + b) \cdot (A \cdot t^2 + 2 \cdot t \cdot v) \leq v^2 + (A + b) \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)$
using ‹$0 \leq A$› ‹$0 < b$› **by** (*smt mult-left-mono*)
also have ... = *?expr1*
by *auto*
finally have $v^2 + (A + b) \cdot (A \cdot t^2 + 2 \cdot t \cdot v) \leq 2 \cdot b \cdot (m - x)$
using *key* **by** *auto*
thus *?thesis*
by (*auto simp*: *field-simps power2-eq-square*)
qed

lemma $A \geq 0 \Longrightarrow b > 0 \Longrightarrow \lceil\lambda s{::}real\hat{~}4.\ s\$2\hat{~}2 \leq 2{*}b{*}(m{-}s\$1) \wedge s\$2 \geq 0\rceil \leq wp$
  $(LOOP$
    $((\lceil\lambda s.\ 2{*}b{*}(m{-}s\$1) \geq s\$2\hat{~}2{+}(A{+}b){*}(A{*}\varepsilon\hat{~}2{+}2{*}\varepsilon{*}(s\$2))\rceil;(3 ::= (\lambda s.\ A))) \cup (3 ::= (\lambda s.\ -b)));$
    $(4 ::= (\lambda s.\ 0));$
    $(x' = (\lambda s.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ (if\ i{=}3\ then\ 0\ else\ 1)))\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$4$
$\leq \varepsilon))$
  $INV\ (\lambda s.\ s\$2\hat{~}2 \leq 2{*}b{*}(m{-}s\$1)))$
  $\lceil\lambda s.\ s\$1 \leq m\rceil$
  apply (*rule wp-loopI*)
    apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*])
   apply(*safe*, *smt not-sum-power2-lt-zero zero-compare-simps*(*10*))
  using *LICSexample4a-arith*[*of A b -\$2 m -\$1 -* ε] **apply** *force*
  by (*auto simp*: *power2-diff power2-eq-square*[*symmetric*] *algebra-simps*(*18*,*19*)
    mult.assoc[*symmetric*] *power-mult-distrib*)

LICS: Example 4b progress of time-triggered car

notation *rel-aka.fdia* ($\diamondsuit$)

lemma *in-fdia-iff-wp*: $(s,s) \in \diamondsuit\ R\ \lceil P\rceil \longleftrightarrow (s,s) \in rel\text{-}ad\ (wp\ R\ (rel\text{-}ad\ \lceil P\rceil))$
  unfolding *rel-ad-def rel-aka.fdia-def rel-aka.fbox-def* **by** (*auto simp*: *p2r-def*)

lemma $\varepsilon > (0{::}real) \Longrightarrow A > 0 \Longrightarrow b > 0 \Longrightarrow$
  $\forall p.\ \exists m.\ (s,s) \in \diamondsuit\ ($
    $LOOP\ (\lceil\lambda s.\ 2{*}b{*}(m{-}x) \geq s\$2\hat{~}2{+}(A{+}b){*}A{*}\varepsilon\hat{~}2{+}2{*}\varepsilon{\cdot}v\rceil;(3 ::= (\lambda s.\ A)) \cup (3 ::= (\lambda s.\ -b)));$
      $(4 ::= (\lambda s.\ 0));$
      $(x' = (\lambda s.\ f\ 1\ (s\$3)\ s)\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$
    $INV\ (\lambda s.\ True))\ \lceil\lambda s.\ s\$1 \geq p\rceil$
  apply(*subst in-fdia-iff-wp*, *simp*)
  apply(*clarsimp simp*: *rel-ad-def*)
  apply(*intro exI conjI allI*)
  apply *clarsimp*
  oops

LICS: Example 4c relative safety of time-triggered car

lemma *in-wp-loopI*:
  $I\ x \Longrightarrow \lceil I\rceil \subseteq \lceil Q\rceil \Longrightarrow \lceil I\rceil \subseteq wp\ R\ \lceil I\rceil \Longrightarrow y = x \Longrightarrow (x,y) \in wp\ (LOOP\ R\ INV\ I)\ \lceil Q\rceil$
  using *wp-loopI*[*of I I Q R*] **apply** *simp*
  apply(*subgoal-tac* $(x,x) \in \lceil I\rceil$)
  by (*simp add*: *subset-eq*, *simp add*: *p2r-def*)

lemma (**in** *local-flow*) *in-wp-g-ode-subset*:
  assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
  shows $(s,s) \in wp\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \lceil Q\rceil \longleftrightarrow\ (s \in S \longrightarrow (\forall t{\in}U\ s.\ (\forall\tau.\ \tau \in U\ s \wedge \tau \leq t$
$\longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
  by (*subst wp-g-ode-subset*[*OF assms*], *simp-all add*: *p2r-def*)

abbreviation *LICS-Ex4c-f* :: $real \Rightarrow real \Rightarrow real\hat{~}4 \Rightarrow real\hat{~}4\ (f)$
  where $f\ time\ acc\ s \equiv (\chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ acc\ else\ if\ i{=}3\ then\ 0\ else\ time))$

lemma *local-flow-LICS-Ex4c-1*:
  $local\text{-}flow\ (f\ k\ a)\ UNIV\ UNIV$
  $(\lambda t\ s.\ \chi\ i.\ if\ i{=}1\ then\ a * t\hat{~}2/2 + s\$2 * t + s\$1\ else$
      $(if\ i{=}2\ then\ a * t + s\$2\ \ \ \ \ \ else$
      $(if\ i{=}3\ then\ s\$3\ \ \ \ \ \ else$
          $k * t + s\$4\ \ \ \ \ )))$

```
  apply(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)
    apply(clarsimp, rule-tac x=1 in exI)+
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def)
  unfolding UNIV-4 by (auto intro!: poly-derivatives simp: forall-4 vec-eq-iff)

lemma local-flow-LICS-Ex4c-2:
  local-flow (λs. f k (s$3) s) UNIV UNIV
  (λt s. χ i. if i=1 then s$3 * t^2/2 + s$2 * t + s$1 else
             (if i=2 then s$3 * t + s$2              else
             (if i=3 then s$3                        else
                          k * t + s$4                )))
  apply(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)
    apply(clarsimp, rule-tac x=1 in exI)+
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def)
  unfolding UNIV-4 by (auto intro!: poly-derivatives simp: forall-4 vec-eq-iff)

lemma LICSexample4c-arith1:
  assumes v^2 ≤ 2 · b · (m − x) 0 ≤ t A ≥ 0 b > 0
    and key: v^2 + (A · (A · ε^2 + 2 · ε · v) + b · (A · ε^2 + 2 · ε · v)) ≤ 2 · b · (m − x)
    and guard: ∀τ. 0 ≤ τ ∧ τ ≤ t ⟶ (0::real) ≤ A · τ + v ∧ τ ≤ ε
  shows (A · t + v)^2 ≤ 2 · b · (m − (A · t^2 / 2 + v · t + x)) (is - ≤ ?rhs)
proof−
  have t ≤ ε 0 ≤ ε 0 ≤ v
    using guard ‹0 ≤ t› by (force, erule-tac x=0 in allE, simp, erule-tac x=0 in allE, simp)
  hence obs1: A · t^2 + 2 · t · v ≤ A · ε^2 + 2 · ε · v
    using ‹A ≥ 0› ‹0 ≤ t› ‹t ≤ ε› by (smt mult-mono power-mono zero-compare-simps(12))
  have obs2:?rhs + A * b * t^2 + 2 * b * v * t = 2 * b * (m − x)
    by (simp add: field-simps)
  have (A · t + v)^2 + A * b * t^2 + 2 * b * v * t = v^2 + (A · (A · t^2 + 2 · t · v) + b · (A · t^2 + 2 · t ·
v))
    by (simp add: field-simps power2-eq-square)
  also have ... ≤ v^2 + (A · (A · ε^2 + 2 · ε · v) + b · (A · ε^2 + 2 · ε · v))
    using obs1 ‹A ≥ 0› ‹b > 0› by (smt mult-less-cancel-left)
  also have ... ≤ 2 * b * (m − x)
    using key .
  finally show ?thesis
    using obs2 by auto
qed

lemma
  assumes A ≥ 0 b > 0 s$2 ≥ 0
  shows (s,s) ∈ wp (x´=(f 0 (−b)) & (λs. True)) ⌈λs. s$1 ≤ m⌉ ⟹
  (s,s) ∈ wp
  (LOOP
    ((⌈λs. 2*b*(m−s$1) ≥ s$2^2+(A+b)*(A*ε^2+2*ε*(s$2))⌉;(3 ::= (λs. A))) ∪ (3 ::= (λs. −b)));
    (4 ::= (λs. 0));
    (x´= (λs. f 1 (s$3) s) & (λs. s$2 ≥ 0 ∧ s$4 ≤ ε))
  INV (λs. s$2^2 ≤ 2*b*(m−s$1))) ⌈λs. s$1 ≤ m⌉
  apply(subst (asm) local-flow.in-wp-g-ode-subset[OF local-flow-LICS-Ex4c-1], simp-all)
  apply(rule in-wp-loopI)
     apply(erule-tac x=s$2/b in allE)
  using ‹b > 0› ‹s$2 ≥ 0› apply(simp add: field-simps power2-eq-square, simp)
    apply (smt ‹b > 0› mult-sign-intros(6) sum-power2-ge-zero)
  apply(simp add: rel-aka.fbox-add2)
   apply(simp-all add: local-flow.wp-g-ode-subset[OF local-flow-LICS-Ex4c-2], safe)
  using LICSexample4c-arith1[OF - - ‹0 ≤ A› ‹0 < b›] apply force
  by (auto simp: field-simps power2-eq-square)
```

no-notation *LICS-Ex4c-f* (f)

LICS: Example 5 Controllability Equivalence

lemma *LICSexample5-arith1*:
  assumes $(0::real) < b\ 0 \leq t$
    and *key*: $v^2 \leq 2 \cdot b \cdot (m - x)$
  shows $v \cdot t - b \cdot t^2 / 2 + x \leq m$
proof−
  have $v^2 \leq 2 \cdot b \cdot (m - x) + (b \cdot t - v)\hat{}2$
    using *key* **by** (*simp add*: *add-increasing2*)
  hence $b\hat{}2 * t\hat{}2 - 2 * b * v * t \geq 2 * b * x - 2 * b * m$
    by (*auto simp*: *field-simps power2-diff*)
  hence $(b\hat{}2/b) * t\hat{}2 - 2 * (b/b) * v * t \geq 2 * (b/b) * x - 2 * (b/b) * m$
    using ‹$b > 0$› **by** (*auto simp*: *field-simps*)
  thus *?thesis*
    using ‹$b > 0$› **by** (*simp add*: *power2-eq-square*)
qed

lemma *LICSexample5-arith2*:
  assumes $(0::real) < b\ 0 \leq v\ \forall t \in \{0..\}.\ v \cdot t - b \cdot t^2 / 2 + x \leq m$
  shows $v^2 \leq 2 \cdot b \cdot (m - x)$
proof(*cases* $v = 0$)
  case *True*
  have $m - x \geq 0$
    using *assms* **by** (*erule-tac* $x=0$ **in** *ballE*, *auto*)
  thus *?thesis*
    using *assms True* **by** *auto*
next
  case *False*
  hence *obs*: $v > 0 \wedge (\exists k.\ k > 0 \wedge v = b * k)$
    using *assms*(1,2) **by** (*metis* (*no-types*, *hide-lams*) *divide-pos-pos divide-self-if*
      less-eq-real-def linorder-not-le mult-1-right mult-1s(1) *times-divide-eq-left*)
  {**fix** $t::real$ **assume** $t \geq 0$
    hence $v \cdot t - b \cdot t^2 / 2 + x \leq m$
      using *assms* **by** *auto*
    hence $-(b\hat{}2) * t\hat{}2 + 2 * b * v * t \leq 2 * b * m - 2 * b * x$
      using ‹$b > 0$› **apply**(*simp add*: *field-simps*)
    by (*metis* (*no-types*, *hide-lams*) *Groups.mult-ac*(1) *nat-distrib*(2) *power2-eq-square real-mult-le-cancel-iff2*)
    hence $v\hat{}2 \leq 2 * b * (m - x) + (b\hat{}2 * t\hat{}2 + v\hat{}2 - 2 * b * v * t)$
      by (*simp add*: *field-simps*)
    also have $... = 2 * b * (m - x) + (b * t - v)\hat{}2$
      by (*simp add*: *power2-diff power-mult-distrib*)
    finally have $v\hat{}2 \leq 2 * b * (m - x) + (b * t - v)\hat{}2$ **.**}
  hence $\forall t \geq 0.\ v^2 \leq 2 \cdot b \cdot (m - x) + (b \cdot t - v)^2$
    by *blast*
  then obtain k **where** $v^2 \leq 2 \cdot b \cdot (m - x) + (b \cdot k - v)^2 \wedge k > 0 \wedge v = b * k$
    using *obs* **by** *fastforce*
  then show *?thesis*
    by *auto*
qed

lemma $b > 0 \Longrightarrow s\$2 \geq 0 \Longrightarrow$
  $(s,s) \in \lceil \lambda s::real\hat{}2.\ s\$2\hat{}2 \leq 2*b*(m-s\$1) \rceil \longleftrightarrow (s,s) \in wp$
  $(x' = (\lambda s.\ \chi\ i.\ if\ i=1\ then\ s\$2\ else\ -b)\ \&\ (\lambda s.\ True))$
  $\lceil \lambda s.\ s\$1 \leq m \rceil$
  apply(*subst local-flow.wp-g-ode-subset*[**where** $T=UNIV$
    and $\varphi=\lambda t\ s.\ \chi\ i::2.\ if\ i=1\ then\ -b * t\hat{}2/2 + s\$2 * t + s\$1\ else\ -b * t + s\$2$])
    apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def forall-2 lipschitz-on-def*)

apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*, *rule-tac x=1* **in** *exI*)+
unfolding *UNIV-2* **apply** *clarsimp*
apply(*force intro*!: *poly-derivatives*)
using *exhaust-2* **apply**(*force simp*: *vec-eq-iff*)
by (*auto simp*: *p2r-def LICSexample5-arith1 LICSexample5-arith2*)

LICS: Example 6 MPC Acceleration Equivalence

lemma *LICSexample6-arith1*:
assumes $0 \leq v$ $0 < b$ $0 \leq A$ $0 \leq \varepsilon$ **and** *guard*: $\forall t\in\{0..\}.\ (\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon) \longrightarrow (\forall \tau\in\{0..\}.$

$A \cdot t \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot t^2 / 2 + v \cdot t + x) \leq (m::real))$
shows $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$
proof−
{**fix** τ::*real*
assume $\tau \geq 0$
hence $A \cdot \varepsilon \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + v \cdot \varepsilon + x) \leq m$
using *guard* ‹$0 \leq \varepsilon$› **apply**(*erule-tac x=ε* **in** *ballE*)
by (*erule impE*, *auto simp*: *closed-segment-eq-real-ivl*)
hence $2 * (A \cdot \varepsilon \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + v \cdot \varepsilon + x)) * b \leq 2 * m * b$
using ‹$0 < b$› **by** (*meson less-eq-real-def mult-left-mono mult-right-mono rel-simps*(*51*))
hence $2 * A \cdot \varepsilon \cdot \tau \cdot b + 2 * v \cdot \tau \cdot b - b\hat{\ }2 \cdot \tau^2 + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$
using ‹$0 < b$› **apply**(*simp add*: *algebra-simps*(*17*,*18*,*19*,*20*) *add.assoc*[*symmetric*]
power2-eq-square[*symmetric*] *mult.assoc*[*symmetric*])
by (*simp add*: *mult.commute mult.left-commute power2-eq-square*)}
hence $\forall \tau\geq 0.\ 2 * A \cdot \varepsilon \cdot \tau \cdot b + 2 * v \cdot \tau \cdot b - b\hat{\ }2 \cdot \tau^2 + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$
by *blast*
moreover have $2 * A \cdot \varepsilon \cdot ((A{*}\varepsilon + v)/b) \cdot b + 2 * v \cdot ((A{*}\varepsilon + v)/b) \cdot b - b\hat{\ }2 \cdot ((A{*}\varepsilon + v)/b)^2 =$
$2 * A \cdot \varepsilon \cdot (A{*}\varepsilon + v) + 2 * v \cdot (A{*}\varepsilon + v) - (A{*}\varepsilon + v)^2$
using ‹$0 < b$› **by** (*simp add*: *field-simps*)
moreover have ... $= v^2 + A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)$
using ‹$0 < b$› **by** (*simp add*: *field-simps power2-eq-square*)
moreover have $(A{*}\varepsilon + v)/b \geq 0$
using *assms* **by** *auto*
ultimately have $v^2 + A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$
using *assms* **by** (*erule-tac x=(A∗ε + v)/b* **in** *allE*, *auto*)
thus *?thesis*
by *argo*
qed

lemma *LICSexample6-arith2*:
assumes $0 \leq v$ $0 < b$ $0 \leq A$ $0 \leq t$ $0 \leq \tau$ $t \leq \varepsilon$
and $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$
shows $A \cdot \varepsilon \cdot \tau + s \mathbin{\$} 2 \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + s \mathbin{\$} 2 \cdot \varepsilon + s \mathbin{\$} 1) \leq m$
sorry

lemma *local-flow-LICS-Ex6*:
local-flow (λ*s*::*real*ˆ*3*. χ *i*. *if i=1 then s\$2 else* (*if i=2 then k1 else k2*)) *UNIV UNIV*
(λ*t s*. χ *i*. *if i* = *1 then k1* ∗ *t*ˆ*2*/*2* + *s\$2* ∗ *t* + *s\$1 else* (*if i=2 then k1* ∗ *t* + *s\$2 else k2* ∗ *t* + *s\$3*))
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def forall-2 lipschitz-on-def*)
apply(*clarsimp*, *rule-tac x=1* **in** *exI*)+
apply(*clarsimp simp*: *dist-norm norm-vec-def L2-set-def*)
unfolding *UNIV-3* **by** (*auto intro*!: *poly-derivatives simp*: *forall-3 vec-eq-iff*)

lemma *s\$2* ≥ *0* ⟹ *b*>*0* ⟹ *A* ≥ *0* ⟹ ε ≥ *0* ⟹
(*s*,*s*) ∈ *wp* ((*3* ::= (λ*s*. *0*));
(*x*´= (λ*s*::*real*ˆ*3*. χ *i*. *if i=1 then s\$2 else* (*if i=2 then A else 1*)) & (λ*s*. *s\$3* ≤ ε)))
(*wp* (*x*´= (λ*s*. χ *i*. *if i=1 then s\$2 else* (*if i=2 then* −*b else 0*)) & (λ*s*. *True*)) ⌈λ*s*. *s\$1* ≤ *m*⌉)
⟷

$(s,s) \in \lceil\lambda s.\ 2*b*(m-s\$1) \geq s\$2\hat{}2+(A+b)*(A*\varepsilon\hat{}2+2*\varepsilon*(s\$2))\rceil$
apply (*simp add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-LICS-Ex6*], *safe*)
apply(*force simp*: *LICSexample6-arith1*)
apply(*erule-tac x=t* **in** *allE*; *clarsimp*)
apply(*rename-tac t* τ)
apply(*rule-tac* $b=A \cdot \varepsilon \cdot \tau + s\ \$\ 2 \cdot \tau - b \cdot \tau^2\ /\ 2 + (A \cdot \varepsilon^2\ /\ 2 + s\ \$\ 2 \cdot \varepsilon + s\ \$\ 1)$ **in** *order.trans*)
apply (*smt divide-le-cancel mult-left-mono mult-right-mono power2-less-imp-less*)
by (*auto simp*: *LICSexample6-arith2*)

LICS: Example 7 Model-Predictive Control Design Car

notation *LICS-Ex4c-f* (*f*)

lemma $s\$2 \geq 0 \Longrightarrow b>0 \Longrightarrow A \geq 0 \Longrightarrow$
$(s,s) \in (wp\ (x'=(f\ 0\ (-b))\ \&\ (\lambda s.\ True)))\ \lceil\lambda s.\ s\$1 \leq m\rceil \Longrightarrow$
$(s,s) \in wp$ (
LOOP
$((\lceil\lambda s.\ (s,s) \in wp\ ((3 ::= (\lambda s.\ 0));(x'= (f\ 1\ A)\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)))$
$(wp\ (x'= (f\ 0\ (-b))\ \&\ (\lambda s.\ True))\ \lceil\lambda s.\ s\$1 \leq m\rceil)\rceil;(3 ::= (\lambda s.\ A))) \cup (3 ::= (\lambda s.\ -b)));$
$(4 ::= (\lambda s.\ 0));\ (x'= (\lambda s.\ f\ 1\ (s\$3)\ s)\ \&\ (\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$
INV $(\lambda s.\ (s,s) \in wp\ (x'=(f\ 0\ (-b))\ \&\ (\lambda s.\ True))\ \lceil\lambda s.\ s\$1 \leq m\rceil))\ \lceil\lambda s.\ s\$1 \leq m\rceil$
oops

no-notation *LICS-Ex4c-f* (*f*)

0.16.2 Advanced

ETCS: Essentials

locale *ETCS* =
fixes ε b A m::*real*

begin

abbreviation *stopDist* $v \equiv v\hat{}2/(2*b)$

abbreviation *accCompensation* $v \equiv ((A/b) + 1) \cdot ((A/2)\cdot\varepsilon\hat{}2 + \varepsilon\cdot v)$

abbreviation *SB* $v \equiv$ *stopDist* v + *accCompensation* v

abbreviation *initial* $m'\ z\ v \equiv (v \geq 0 \wedge (m' - z) \geq$ *stopDist* $v)$

abbreviation *safe* $m'\ z\ v\ \delta \equiv z \geq m' \longrightarrow v \leq \delta$

abbreviation *loopInv* $m'\ z\ v \equiv v \geq 0 \wedge m' - z \geq$ *stopDist* v

abbreviation *ctrl* $\equiv \lceil\lambda s$::*real*$\hat{}4.\ m - s\$1 \leq SB\ (s\$2)\rceil;(3 ::= (\lambda s.\ -b)) \cup$
$\lceil\lambda s.\ m - s\$1 \geq SB\ (s\$2)\rceil;(3 ::= (\lambda s.\ A))$

abbreviation *drive* $\equiv (4 ::= (\lambda s.\ 0));$
$(x'= (\lambda s$::*real*$\hat{}4.\ \chi\ i.\ if\ i{=}1\ then\ s\$2\ else\ (if\ i{=}2\ then\ s\$3\ else\ (if\ i{=}3\ then\ 0\ else\ 1))))$

& ($\lambda s.\ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon$))

lemma *ETCS-arith1*:
assumes $0 < b$ $0 \leq A$ $0 \leq v$ $0 \leq t$
and $v^2 / (2 \cdot b) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v)/ b + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v)) \leq m - x$ (**is** *?expr1* $\leq m - x$)
and *guard*: $\forall \tau.\ 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon$
shows $(A \cdot t + v)^2/(2 \cdot b) \leq m - (A \cdot t^2/2 + v \cdot t + x)$ (**is** *?lhs* $\leq$ *?rhs*)
proof−
have $2 \cdot b \cdot (v^2/(2 \cdot b) + (A \cdot (A \cdot \varepsilon^2/2+\varepsilon \cdot v)/b + (A \cdot \varepsilon^2/2+\varepsilon \cdot v))) \leq 2 \cdot b \cdot (m-x)$ (**is** *?expr2* $\leq 2 \cdot b \cdot (m-x)$)
using ‹$0 < b$› *mult-left-mono*[*OF* ‹*?expr1* $\leq m - x$›, *of* $2 \cdot b$] **by** *auto*
also have *?expr2* $= v^2 + 2 \cdot A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v$
using ‹$0 < b$› **by** (*auto simp*: *field-simps*)
also have ... $= v^2 + A\hat{\ }2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v$
by (*auto simp*: *field-simps power2-eq-square*)
finally have *obs*: $v^2 + A^2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v \leq 2 \cdot b \cdot (m-x)$ (**is** *?expr3* $\varepsilon \leq 2 \cdot b \cdot (m-x)$) **.**
have $t \leq \varepsilon$
using *guard* ‹$0 \leq t$› **by** *auto*
hence $v^2 + A^2 \cdot t^2 + b \cdot A \cdot t^2 \leq v^2 + A^2 \cdot \varepsilon^2 + b \cdot A \cdot \varepsilon^2$
using *power-mono*[*OF* ‹$t \leq \varepsilon$› ‹$0 \leq t$›, *of* 2]
by (*smt assms*(*1*,*2*) *mult-less-cancel-left zero-compare-simps*(*4*) *zero-le-power*)
hence $v^2 + A^2 \cdot t^2 + 2 \cdot A \cdot t \cdot v + b \cdot A \cdot t^2 \leq v^2 + A^2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2$
using *assms*(*1*,*2*,*3*,*4*) ‹$t \leq \varepsilon$› **by** (*smt mult-left-mono mult-right-mono*)
hence *?expr3* $t \leq 2 \cdot b \cdot (m - x)$
using *assms*(*1*,*2*,*3*,*4*) ‹$t \leq \varepsilon$› *obs* **by** (*smt mult-right-mono real-mult-le-cancel-iff2*)
hence $A^2 \cdot t^2 + v^2 + 2 \cdot A \cdot t \cdot v \leq 2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x$
by (*simp add*: *right-diff-distrib*)
hence $(A \cdot t + v)^2 \leq 2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x$
unfolding *cross3-simps*(*29*)[*of A t 2*] *power2-sum*[*of A·t v*] **by** (*simp add*: *mult.assoc*)
hence *?lhs* $\leq (2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x)/(2 \cdot b)$ (**is** - $\leq$ *?expr4*)
using ‹$0 < b$› *divide-right-mono* **by** *fastforce*
also have *?expr4* = *?rhs*
using ‹$0 < b$› **by** (*auto simp*: *field-simps*)
finally show *?lhs* $\leq$ *?rhs* **.**
qed

lemma $b > 0 \Longrightarrow A \geq 0 \Longrightarrow \varepsilon \geq 0 \Longrightarrow$
$\lceil\lambda s.$ *initial m* (*s\$1*) (*s\$2*)$\rceil \leq$ *wp* (*LOOP ctrl*;*drive INV* ($\lambda s.$ *loopInv m* (*s\$1*) (*s\$2*))) $\lceil\lambda s.\ s\$1 \leq m\rceil$
apply (*rule wp-loopI*)
apply (*simp-all add*: *le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*], *safe*)
apply (*smt divide-le-cancel divide-minus-left not-sum-power2-lt-zero*)
apply(*auto simp*: *field-simps power2-eq-square*)[*1*]
using *ETCS-arith1* **by** *force*

end

end

0.17 Verification components with Kleene Algebras

We create verification rules based on various Kleene Algebras.

theory *VC-diffKAD-KA*
imports
KAT-and-DRA.PHL-KAT
KAD.Modal-Kleene-Algebra
Transformer-Semantics.Kleisli-Quantale

begin

0.17.1 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**
$H\ p\ x\ q \longleftrightarrow \mathsf{tt}\ p \cdot x \leq x \cdot \mathsf{tt}\ q$

lemma *H-consl*: $\mathsf{tt}\ p \leq \mathsf{tt}\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$
  using *Hoare-def phl-cons1* **by** *blast*

lemma *H-consr*: $\mathsf{tt}\ q' \leq \mathsf{tt}\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$
  using *Hoare-def phl-cons2* **by** *blast*

lemma *H-cons*: $\mathsf{tt}\ p \leq \mathsf{tt}\ p' \Longrightarrow \mathsf{tt}\ q' \leq \mathsf{tt}\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$
  by (*simp add*: *H-consl H-consr*)

— Skip program

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

— Sequential composition

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

— Conditional statement

definition *kat-cond* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*if - then - else - fi* [*64*,*64*,*64*] *63*) **where**
if p then x else y fi $= (\mathsf{tt}\ p \cdot x + n\ p \cdot y)$

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

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

lemma *H-cond*: $H\ (\mathsf{tt}\ p \cdot \mathsf{tt}\ r)\ x\ q \Longrightarrow H\ (\mathsf{tt}\ p \cdot n\ r)\ y\ q \Longrightarrow H\ p$ (*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 = $(\mathsf{tt}\ b \cdot x)^\star \cdot n\ b$

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

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

— Invariants

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

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

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

end

0.17.2 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*: $tt\ p \leq tt\ p' \Longrightarrow tt\ q' \leq tt\ q \Longrightarrow Ref\ p'\ q' \leq Ref\ p\ q$
proof −
  assume *h1*: $tt\ p \leq tt\ p'$ **and** *h2*: $tt\ q' \leq tt\ q$
  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*) · (*Ref r q*)) *q*
by (*rule H-seq*)
thus *?thesis*
by (*rule R2*)
qed

— Conditional statement

lemma *R-cond*: *if v then* (*Ref* (tt *v* · tt *p*) *q*) *else* (*Ref* (*n v* · tt *p*) *q*) *fi* ≤ *Ref p q*
proof −
have *H* (tt *v* · tt *p*) (*Ref* (tt *v* · tt *p*) *q*) *q* **and** *H* (*n v* · tt *p*) (*Ref* (*n v* · tt *p*) *q*) *q*
by (*simp add*: *R1*)+
hence *H p* (*if v then* (*Ref* (tt *v* · tt *p*) *q*) *else* (*Ref* (*n v* · tt *p*) *q*) *fi*) *q*
by (*simp add*: *H-cond n-mult-comm*)
thus *?thesis*
by (*rule R2*)
qed

— While loop

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

— Finite iteration

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

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

— Invariants

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

end

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

0.17.3 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 $aka\text{-}cond :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (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 $aka\text{-}loop\text{-}inv :: 'a \Rightarrow 'a \Rightarrow 'a$ ($loop$ - inv - [64,64] 63)
  where $loop\ x\ inv\ i = x^\star$

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

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

— Invariants

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

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

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

end

0.17.4 Relational model

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

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

lemma *power-is-relpow*: $rel\text{-}uq.power\ X\ m = X\ \hat{}\hat{}\ m$ **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*)

0.17.5 State transformer model

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

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

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

lemma *nd-fun-ext*: $(\bigwedge x.\ (f_\bullet)\ x = (g_\bullet)\ x) \Longrightarrow f = g$
  apply(*subgoal-tac Rep-nd-fun f = Rep-nd-fun g*)
  using *Rep-nd-fun-inject*

apply *blast*
by(*rule ext*, *simp*)

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

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

definition $0 = \zeta^\bullet$

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

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

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

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

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

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

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

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

end

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

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

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

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

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

end

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

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

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

end

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

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

lemma *nd-fun-ad-zero*[*nd-fun-aka*]: $ad\ x \cdot x = 0$
and *nd-fun-ad*[*nd-fun-aka*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$
and *nd-fun-ad-one*[*nd-fun-aka*]: $ad\ (ad\ x) + ad\ x = 1$ **for** x::$'a$ *nd-fun*
unfolding *antidomain-op-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*
by (*auto simp*: *nd-fun-eq-iff kcomp-def one-nd-fun-def*)

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

end

end

0.18 VC_diffKAD

theory *VC-diffKAD-auxiliarities*
imports *Main VC-diffKAD-KA Ordinary-Differential-Equations.ODE-Analysis*

begin

0.18.1 Stack Theories Preliminaries: VC_KAD and ODEs

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

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

type-synonym $'a$ *store* = *string* $\Rightarrow$ $'a$

hide-const η

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)
and *antidomain-semiringl.ads-d* (d)

and *n-op* (n - [*90*] *91*)
and *Hoare* (H)
and *tau* (τ)
and *dual* (∂)
and *fres* (**infixl** $\leftarrow$ *60*)
and *n-add-op* (**infixl** $\oplus$ *65*)
and *eta* (η)

notation *Set.image* (-(|-|))
and *Product-Type.prod.fst* (π_1)
and *Product-Type.prod.snd* (π_2)
and *List.zip* (**infixl** $\otimes$ *63*)
and *rel-aka.fbox* (*wp*)

definition *p2r* :: *'a pred* $\Rightarrow$ *'a rel* ((*1*$\lceil$-$\rceil$)) **where**
$\lceil P\rceil = \{(s,s) \mid s.\ P\ s\}$

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

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

lemma *boxProgrPred-chrctrztn*: $(x,y) \in$ *wp* $R\ \lceil P\rceil \longleftrightarrow (y = x \wedge (\forall y.\ (x,\ y) \in R \longrightarrow P\ y))$
unfolding *wp-rel* **unfolding** *p2r-def* **by** *simp*

definition *assign* :: *string* $\Rightarrow$ (*'a store* $\Rightarrow$ *'a*) $\Rightarrow$ (*'a store*) *rel* (- ::= -)
where (x ::= e) = $\{(s,\ s(x := e\ s)) \mid s.$ *True* $\}$

lemma *wp-assign* [*simp*]: *wp* (x ::= e) $\lceil P\rceil = \lceil\lambda\ s.\ P\ (s(x := e\ s))\rceil$
unfolding *wp-rel assign-def* **by** *simp*

abbreviation *cond-sugar* :: *'a pred* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a rel* (*IF* - *THEN* - *ELSE* - [*64*,*64*] *63*)
where *IF P THEN X ELSE Y* $\equiv$ *rel-aka.aka-cond* $\lceil P\rceil$ X Y

lemma *wp-loopI*: $\lceil P\rceil \le \lceil I\rceil \Longrightarrow \lceil I\rceil \le \lceil Q\rceil \Longrightarrow \lceil I\rceil \le$ *wp* $R\ \lceil I\rceil \Longrightarrow \lceil P\rceil \le$ *wp* $(R^*)\ \lceil Q\rceil$
using *rel-aka.fbox-stari*[*of* $\lceil P\rceil$ $\lceil I\rceil$] **by** *auto*

proposition *cons-eq-zipE*:
$(x,\ y)$ # *tail* = *xList* $\otimes$ *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*]

0.18.2 VC_diffKAD Preliminaries

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

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

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

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

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

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

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

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

(primed) dSolve preliminaries

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

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

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

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

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

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

primrec *state-list-upd* :: ((*real* ⇒ *real store* ⇒ *real*) × *string* × (*real store* ⇒ *real*)) *list* ⇒
real ⇒ *real store* ⇒ *real store* **where**
  state-list-upd [] *t s* = *s*|
  state-list-upd (*uxf* # *tail*) *t s* = (*state-list-upd tail t s*)
    ((π_1 (π_2 *uxf*)) := (π_1 *uxf*) *t s*,
      ∂ (π_1 (π_2 *uxf*)) := (*if t* = *0 then* (π_2 (π_2 *uxf*)) *s*
        else vderiv-of (λ *r*. (π_1 *uxf*) *r s*) {*0*<..< (*2* $*_R$ *t*)} *t*))

abbreviation *state-list-cross-upd* ::*real store* ⇒ (*string* × (*real store* ⇒ *real*)) *list* ⇒
(*real* ⇒ *real store* ⇒ *real*) *list* ⇒ *real* ⇒ (*char list* ⇒ *real*) (-[-←-] - [*64*,*64*,*64*] *63*)

where $s[xfList \leftarrow uInput]\ t \equiv state\text{-}list\text{-}upd\ (uInput \otimes xfList)\ t\ s$

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

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

theorem *state-list-cross-upd-its-vars*:
assumes *distinctHyp*:$distinct\ (map\ \pi_1\ xfList)$
and *lengthHyp*:$length\ xfList = length\ uInput$
and *varsHyp*:$\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *its-var*: $(u,x,f) \in set\ (uInput \otimes xfList)$
shows $(s[xfList \leftarrow uInput]\ t)\ x = u\ t\ s$
using *assms* **apply**(*induct xfList uInput arbitrary*: *x rule*: *list-induct2'*, *simp*, *simp*, *simp*)
by(*clarify*, *rule inductive-state-list-cross-upd-its-vars*, *simp-all*)

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

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

ultimately show *?thesis* **by** *auto*
qed

theorem *state-list-cross-upd-its-dvars*:
assumes *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0 s* = *s* (π_1 (π_2 *uxf*))
shows $\exists$ *g*. (*s*[*xfList*$\leftarrow$*uInput*] *0*) = (*override-on s g varDiffs*)
using *assms* **proof**(*induct xfList uInput rule*: *list-induct2'*)
case *1*
have (*s*[[]$\leftarrow$[]] *0*) = *override-on s s varDiffs*
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *metis*
next
case (*2 xf xfTail*)
have (*s*[(*xf* # *xfTail*)$\leftarrow$[]] *0*) = *override-on s s varDiffs*
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *metis*
next
case (*3 u utail*)
have (*s*[[]$\leftarrow$*utail*] *0*) = *override-on s s varDiffs*
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *force*
next
case (*4 xf xfTail u uTail*)
then have $\exists$ *g*. (*s*[*xfTail*$\leftarrow$*uTail*] *0*) = *override-on s g varDiffs* **by** *simp*
thus *?case* **using** *inductive-state-list-cross-upd-its-dvars 4.prems* **by** *blast*
qed

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

lemma *has-vderiv-on-cong-open-interval*:
assumes *gHyp*:$\forall$ τ > *0*. *f* τ = *g* τ **and** *tHyp*: *t*>*0*
and *fHyp*:(*f has-vderiv-on f'*) {*0*<..<*t*}
shows (*g has-vderiv-on f'*) {*0*<..<*t*}
proof−
from *gHyp* **have** $\bigwedge\tau$. $\tau \in$ {*0*<..< *t*} $\Longrightarrow$ *f* τ = *g* τ **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*:$\forall$ τ > *0*. *f* τ = *g* τ
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'* τ = *f''* τ)
using *vderiv-unique-within-open-interval ctHyp* **by** *blast*

also have (g has-vderiv-on (vderiv-of g {0<..< (c $*_R$ t)})) {0<..<c · t}
by(simp add: vderiv-of-def, metis derivHyp someI-ex)
ultimately showvderiv-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:∀ xf ∈ set xfList. π_1 xf ∉ varDiffs
and solHyp2:∀ t≥0. ((λτ. (sol s[xfList←uInput] τ) x)
has-vderiv-on (λτ. f (sol s[xfList←uInput] τ))) {0..t}
and tHyp: t>0 **and** uxfHyp:(u, x, f) ∈ set (uInput ⊗ xfList)
shows vderiv-of (λτ. u τ (sol s)) {0<..< (2 $*_R$ t)} t = f (sol s[xfList←uInput] t)
apply(rule-tac f=(λτ. (sol s[xfList←uInput] τ) x) **in** closed-vderiv-on-cong-to-open-vderiv)
subgoal using assms **and** state-list-cross-upd-its-vars **by** metis
by(simp-all add: solHyp2 tHyp)

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

lemma to-sol-zero-its-dvars:
assumes funcsHyp:∀ s. ∀ g. ∀ xf ∈ set xfList. π_2 xf (override-on s g varDiffs) = π_2 xf s
and distinctHyp:distinct (map π_1 xfList)
and lengthHyp:length xfList = length uInput
and varsHyp:∀ xf ∈ set xfList. π_1 xf ∉ varDiffs
and solHyp1:∀ uxf ∈ set (uInput ⊗ xfList). (π_1 uxf) 0 (sol s) = (sol s) (π_1 (π_2 uxf))
and ygHyp:(y, g) ∈ set xfList
shows (sol s[xfList←uInput] 0)(∂ y) = g (sol s[xfList←uInput] 0)

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

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

lemma *to-sol-greater-than-zero-its-dvars*:
assumes *distinctHyp*:$distinct\ (map\ \pi_1\ xfList)$
and *lengthHyp*:$length\ xfList = length\ uInput$
and *varsHyp*:$\forall xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *uxfHyp*:$(u,\ x,\ f) \in set\ (uInput \otimes xfList)$ **and** *tHyp*:$t > 0$
shows $(s[xfList{\leftarrow}uInput]\ t)\ (\partial\ x) = vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0{<}..{<}\ (2\ {*_R}\ t)\}\ t$
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*, *clarify*)
by(*rule-tac f=f* **in** *inductive-to-sol-greater-than-zero-its-dvars*, *auto*)

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*)

term *test-monoid-class.n-add-op*

primrec *tval* ::*trms* $\Rightarrow$ (*real store* $\Rightarrow$ *real*) (($1[\![\ \text{-}\ ]\!]_t$)) **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**
$[\![\vartheta \doteq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s = ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \prec \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s < ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \preceq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s \le ([\![\eta]\!]_t)\ s)|$
$[\![\varphi \sqcap \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \wedge ([\![\psi]\!]_P)\ s)|$
$[\![\varphi \sqcup \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \vee ([\![\psi]\!]_P)\ s)$

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* π_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* π_1 ((*map* (*vdiff* $\circ$ π_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* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)$\langle\eta\rangle = \eta$
using *assms substList-help1 substList-on-compl-of-varDiffs* **by** *blast*

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

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

primrec *subspList* :: (*string* $\times$ *trms*) *list* $\Rightarrow$ *props* $\Rightarrow$ *props* (-$\restriction$-$\restriction$ [*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*:
((λt::*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*(*16*))
by (*auto intro*: *derivative-eq-intros*)

lemma *quadratic-monomial-derivative2*:
((λt::*real*. $a \cdot t^2$ / *2*) *has-derivative* ($\lambda t.\ a \cdot x \cdot t$)) (*at x within T*)

apply(*rule-tac f'1*=λ*t. a* · (*2* · *x* · *t*) **and** *g'1*=λ *x. 0* **in** *derivative-eq-intros*(*18*))
using *quadratic-monomial-derivative* **by** *auto*

lemma *quadratic-monomial-vderiv*[*poly-deriv*]:((λ*t. a* · t^2 / *2*) *has-vderiv-on* (·) *a*) *T*
apply(*simp add*: *has-vderiv-on-def has-vector-derivative-def*, *clarify*)
using *quadratic-monomial-derivative2* **by** (*simp add*: *mult-commute-abs*)

lemma *galilean-position*[*galilean-transform*]:
((λ*t. a* · t^2 / *2* + *v* · *t* + *x*) *has-vderiv-on* (λ*t. a* · *t* + *v*)) *T*
apply(*rule-tac f'*=λ *x. a* · *x* + *v* **and** *g'1*=λ *x. 0* **in** *derivative-intros*(*189*))
apply(*rule-tac f'1*=λ *x. a* · *x* **and** *g'1*=λ *x. v* **in** *derivative-intros*(*189*))
using *poly-deriv*(*2*) **by**(*auto intro*: *derivative-intros*)

lemma [*poly-deriv*]:
t ∈ *T* ⟹ ((λτ. *a* · τ^2 / *2* + *v* · τ + *x*) *has-derivative* (λ*x. x* $*_R$ (*a* · *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* ⟹ *vderiv-of* (λ*t. a* · *t*ˆ*2* / *2* + *v* · *t* + *x*) {*0*<..<*2* · *t*} *t* = *a* · *t* + *v*
proof−
let *?f* = *vderiv-of* (λ*t. a* · *t*ˆ*2* / *2* + *v* · *t* + *x*) {*0*<..<*2* · *t*}
assume *t* > *0* **hence** *t* ∈ {*0*<..<*2* · *t*} **by** *auto*
have ∃ *f*. ((λ*t. a* · t^2 / *2* + *v* · *t* + *x*) *has-vderiv-on f*) {*0*<..<*2* · *t*}
using *galilean-position* **by** *blast*
hence ((λ*t. a* · *t*ˆ*2* / *2* + *v* · *t* + *x*) *has-vderiv-on ?f*) {*0*<..<*2* · *t*}
unfolding *vderiv-of-def* **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)
also have ((λ*t. a* · t^2 / *2* + *v* · *t* + *x*) *has-vderiv-on* (λ*t. a* · *t* + *v*)) {*0*<..<*2* · *t*}
using *galilean-position* **by** *simp*
ultimately show (*vderiv-of* (λ*t. a* · *t*ˆ*2* / *2* + *v* · *t* + *x*) {*0*<..<*2* · *t*}) *t* = *a* · *t* + *v*
apply(*rule-tac f'*=*?f* **and** τ=*t* **and** *t*=*2*·*t* **in** *vderiv-unique-within-open-interval*)
using ‹*t* ∈ {*0*<..<*2* · *t*}› **by** *auto*
qed

lemma *t* > *0* ⟹ *vderiv-of* (λ*t. a* · *t*ˆ*2* / *2* + *v* · *t* + *x*) {*0*<..<*2* · *t*} *t* = *a* · *t* + *v*
unfolding *vderiv-of-def* **apply**(*subst some1-equality*[*of* - (λ*t. a* · *t* + *v*)])
apply(*rule-tac a*=λ*t. a* · *t* + *v* **in** *ex1I*)
apply(*simp-all add*: *galilean-position*)
apply(*rule ext*, *rename-tac f* τ)
apply(*rule-tac f*=λ*t. a* · t^2 / *2* + *v* · *t* + *x* **and** *t*=*2* · *t* **and** *f'*=*f* **in** *vderiv-unique-within-open-interval*)
apply(*simp-all add*: *galilean-position*)
oops

lemma *galilean-velocity*[*galilean-transform*]:((λ*r. a* · *r* + *v*) *has-vderiv-on* (λ*t. a*)) *T*
apply(*rule-tac f'1*=λ *x. a* **and** *g'1*=λ *x. 0* **in** *derivative-intros*(*189*))
unfolding *has-vderiv-on-def* **by**(*auto intro*: *derivative-eq-intros*)

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

using ‹$t \in \{0<..<2 \cdot t\}$› **by** *auto*
qed

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

lemma [*galilean-transform-eq*]:$t > 0 \Longrightarrow \textit{vderiv-of}\ (\lambda t.\ v \cdot t - a \cdot t\hat{}2\ /\ 2 + x)\ \{0<..<2 \cdot t\}\ t = v - a \cdot t$
apply(*subgoal-tac* $\textit{vderiv-of}\ (\lambda t.\ - a \cdot t^2\ /\ 2 + v \cdot t + x)\ \{0<..<2 \cdot t\}\ t = - a \cdot t + v$, *simp*)
by(*rule galilean-transform-eq*)

lemma [*galilean-transform*]:
$((\lambda t.\ v - a \cdot t)\ \textit{has-vderiv-on}\ (\lambda x.\ - a))\ \{0..t\}$
apply(*subgoal-tac* $((\lambda t.\ - a \cdot t + v)\ \textit{has-vderiv-on}\ (\lambda x.\ - a))\ \{0..t\}$, *simp*)
by(*rule galilean-transform*)

lemma [*galilean-transform-eq*]:$t > 0 \Longrightarrow \textit{vderiv-of}\ (\lambda r.\ v - a \cdot r)\ \{0<..<2 \cdot t\}\ t = - a$
apply(*subgoal-tac* $\textit{vderiv-of}\ (\lambda t.\ - a \cdot t + v)\ \{0<..<2 \cdot t\}\ t = - a$, *simp*)
by(*rule galilean-transform-eq*)

lemma [*simp*]:$(\lambda x.\ \textit{case}\ x\ \textit{of}\ (t, x) \Rightarrow f\ t) = (\lambda\ x.\ (f \circ \pi_1)\ x)$
by *auto*

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

begin

0.18.3 Phase Space Relational Semantics

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

lemma *solves-store-ivpI*:
assumes $\forall\ t \geq 0.\forall\ xf \in \textit{set xfList}.\ (\varphi_S\ t\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (\varphi_S\ t)$
and $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|\textit{set xfList}|\!)) \cup \textit{varDiffs} \longrightarrow \varphi_S\ t\ y = s\ y$
and $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|\textit{set xfList}|\!)) \longrightarrow \varphi_S\ t\ (\partial\ y) = 0$
and $\forall\ t \geq 0.\ \forall\ xf \in \textit{set xfList}.\ ((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))\ \textit{solves-ode}\ (\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}\ \textit{UNIV}$
and $\forall\ xf \in \textit{set xfList}.\ \varphi_S\ 0\ (\pi_1\ xf) = s(\pi_1\ xf)$
shows φ_S *solvesTheStoreIVP xfList withInitState s*
apply(*simp add*: *solvesStoreIVP-def*, *safe*)
using *assms* **apply** *simp-all*
by(*force,force,force*)

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

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*

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

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
shows $\forall\ y.\ y \notin varDiffs \longrightarrow \varphi_S\ 0\ y = s\ y$
proof(*clarify*, *rename-tac x*)
fix *x* **assume** $x \notin varDiffs$
from *assms* **and** *solves-store-ivpE(5)* **have** $x \in (\pi_1 (\!| set\ xfList |\!)) \Longrightarrow \varphi_S\ 0\ x = s\ x$ **by** *fastforce*
also have $x \notin (\pi_1 (\!| set\ xfList |\!)) \cup varDiffs \Longrightarrow \varphi_S\ 0\ x = s\ x$
using *assms* **and** *solves-store-ivpE(1)* **by** *simp*
ultimately show $\varphi_S\ 0\ x = s\ x$ **using** ‹$x \notin varDiffs$› **by** *auto*
qed

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

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

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

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

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$
and $xf \in set\ xfList$
shows $((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))\ solves\text{-}ode\ (\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}\ UNIV$
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* :: $(string \times (real\ store \Rightarrow real))\ list \Rightarrow (real\ store\ pred) \Rightarrow$
$real\ store\ rel\ (ODEsystem\ \text{-}\ with\ \text{-}\ \ [70,\ 70]\ 61)$
  where $ODEsystem\ xfList\ with\ G =$
    $\{(s,\varphi_S\ t)\ |s\ t\ \varphi_S.\ t \geq 0 \wedge (\forall\ \ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge solvesStoreIVP\ \varphi_S\ xfList\ s\}$

abbreviation *vericond* :: $'a\ pred \Rightarrow\ 'a\ rel \Rightarrow\ 'a\ pred \Rightarrow bool\ (PRE\ \text{-}\ \text{-}\ POST\ \text{-})$
  where $PRE\ P\ X\ POST\ Q \equiv \lceil P \rceil \subseteq wp\ X\ \lceil Q \rceil$

lemma *vericond-gevol*: $(PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ Q) =$
  $(\forall s.\ P\ s \longrightarrow (\forall s'.\ (s,s') \in (ODEsystem\ xfList\ with\ G) \longrightarrow Q\ s'))$
  unfolding *wp-rel* **by** *clarsimp*

0.18.4 Derivation of Differential Dynamic Logic Rules

"Differential Weakening"

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

theorem *dWeakening*:
  assumes *guardImpliesPost*: $\lceil G \rceil \subseteq \lceil Q \rceil$
  shows $PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ Q$
  unfolding *wp-rel guarDiffEqtn-def* **using** *assms* **by** *auto*

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

"Differential Cut"

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

theorem *dCut*:
  assumes *pBoxDiffCut*:$(PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ C)$
  assumes *pBoxCutQ*:$(PRE\ P\ (ODEsystem\ xfList\ with\ (\lambda\ s.\ G\ s \wedge C\ s))\ POST\ Q)$
  shows $PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ Q$
proof(*clarsimp simp*: *wp-rel*)
  fix $b\ y$ **assume** $P\ b$ **and** $(b,\ y) \in ODEsystem\ xfList\ with\ G$
  then obtain $\varphi_S\ t$ **where** $*$:$solvesStoreIVP\ \varphi_S\ xfList\ b \wedge (\forall\ \ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge 0 \leq t \wedge \varphi_S\ t = y$
    using *guarDiffEqtn-def* **by** *auto*
  hence $\forall\ \ r \in \{0..t\}.\ (b,\ \varphi_S\ r) \in (ODEsystem\ xfList\ with\ G)$
    using *all-interval-guarDiffEqtn* **by** *blast*
  from *this* **and** *pBoxDiffCut* **have** $\forall\ \ r \in \{0..t\}.\ C\ (\varphi_S\ r)$

```
    using ‹P b› by(clarsimp simp: wp-rel)
  then have ∀ r ∈ {0..t}. (b, φS r) ∈ (ODEsystem xfList with (λ s. G s ∧ C s))
    using ∗ all-interval-guarDiffEqtn by (metis (mono-tags, lifting))
  from this and pBoxCutQ have ∀ r ∈ {0..t}. Q (φS r)
    using boxProgrPred-chrctrztn ‹P b› by(clarsimp simp: wp-rel)
  thus Q y
    using ∗ by auto
qed
```

```
theorem dC:
  assumes Id ⊆ wp (ODEsystem xfList with G) ⌈C⌉
  shows wp (ODEsystem xfList with G ) ⌈Q⌉ = wp (ODEsystem xfList with (λ s. G s ∧ C s)) ⌈Q⌉
proof(rule-tac f=λ x. wp x ⌈Q⌉ in HOL.arg-cong, safe)
  fix a b assume (a, b) ∈ ODEsystem xfList with G
  then obtain φS t where ∗:solvesStoreIVP φS xfList a ∧ (∀ r ∈ {0..t}. G (φS r)) ∧ 0 ≤ t ∧ φS t = b
    using guarDiffEqtn-def by auto
  hence 1:∀ r ∈ {0..t}. (a, φS r) ∈ ODEsystem xfList with G
    by (meson all-interval-guarDiffEqtn)
  from this have ∀ r ∈ {0..t}. C (φS r)
    using assms boxProgrPred-chrctrztn
    by (metis (no-types, hide-lams) basic-trans-rules(31) pair-in-Id-conv)
  thus (a, b) ∈ ODEsystem xfList with (λs. G s ∧ C s)
    using ∗ guarDiffEqtn-def by blast
next
  fix a b assume (a, b) ∈ ODEsystem xfList with (λs. G s ∧ C s)
  then show (a, b) ∈ ODEsystem xfList with G
    unfolding guarDiffEqtn-def by(clarsimp, rule-tac x=t in exI, rule-tac x=φS in exI, simp)
qed
```

Solve Differential Equation

```
lemma prelim-dSolve:
  assumes solHyp:(λt. sol s[xfList←uInput] t) solvesTheStoreIVP xfList withInitState s
    and uniqHyp:∀ X. solvesStoreIVP X xfList s ⟶ (∀ t ≥ 0. (sol s[xfList←uInput] t) = X t)
    and diffAssgn: ∀t≥0. G (sol s[xfList←uInput] t) ⟶ Q (sol s[xfList←uInput] t)
  shows ∀ c. (s,c) ∈ (ODEsystem xfList with G) ⟶ Q c
proof(clarify)
  fix c assume (s,c) ∈ (ODEsystem xfList with G)
  from this obtain t::real and φS::real ⇒ real store
    where FHyp:t≥0 ∧ φS t = c ∧ solvesStoreIVP φS xfList s ∧ (∀ r ∈ {0..t}. G (φS r))
    using guarDiffEqtn-def by auto
  from this and uniqHyp have (sol s[xfList←uInput] t) = φS t by blast
  then have cHyp:c = (sol s[xfList←uInput] t) using FHyp by simp
  from this have G (sol s[xfList←uInput] t) using FHyp by force
  then show Q c using diffAssgn FHyp cHyp by auto
qed
```

```
theorem dS:
  assumes solHyp:∀ s. solvesStoreIVP (λt. sol s[xfList←uInput] t) xfList s
    and uniqHyp:∀ s X. solvesStoreIVP X xfList s ⟶ (∀ t ≥ 0. (sol s[xfList←uInput] t) = X t)
  shows wp (ODEsystem xfList with G) ⌈Q⌉ =
  ⌈λ s. ∀t≥0. (∀r∈{0..t}. G (sol s[xfList←uInput] r)) ⟶ Q (sol s[xfList←uInput] t)⌉
  apply(simp add: p2r-def, rule subset-antisym)
  unfolding guarDiffEqtn-def rel-aka.fbox-def rel-ad-def
  using solHyp apply(simp add: relcomp.simps) apply clarify
   apply(rule-tac x=x in exI, clarsimp)
   apply(erule-tac x=sol x[xfList←uInput] t in allE, erule disjE)
    apply(erule-tac x=x in allE, erule-tac x=t in allE)
    apply(erule impE, simp, erule-tac x=λt. sol x[xfList←uInput] t in allE)
    apply(simp-all, clarify, rule-tac x=s in exI, simp add: relcomp.simps)
```

using *uniqHyp* **by** *fastforce*

theorem *dSolve*:
assumes *solHyp*:$\forall s.$ *solvesStoreIVP* $(\lambda t.$ *sol s[xfList←uInput] t*$)$ *xfList s*
and *uniqHyp*:$\forall$ *s*. $\forall$ *X*. *solvesStoreIVP X xfList s* $\longrightarrow$ $(\forall\ t \geq 0.$(*sol s[xfList←uInput] t*$) = X\ t)$
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(*clarsimp*, *subgoal-tac a=b*)
apply(*clarify*, *subst boxProgrPred-chrctrztn*)
apply(*simp-all add*: *p2r-def*)
apply(*rule-tac uInput=uInput* **in** *prelim-dSolve*)
apply(*simp add*: *solHyp*, *simp add*: *uniqHyp*)
by (*metis* (*no-types*, *lifting*) *diffAssgn*)

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

lemma *conds4vdiffs-prelim*:
assumes *funcsHyp*:$\forall s\ g.\ \forall xf{\in}$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) $= \pi_2$ *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:$\forall$ *uxf* $\in$ *set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall t{\geq}0$. (($\lambda\tau$. (*sol s[xfList←uInput]* τ) *x*)
has-vderiv-on ($\lambda\tau$. *f* (*sol s[xfList←uInput]* τ))) $\{0..t\}$
and *xfHyp*:(*x*, *f*) $\in$ *set xfList* **and** *tHyp*:$t \geq 0$
shows (*sol s[xfList←uInput] t*) (∂ *x*) = *f* (*sol s[xfList←uInput] t*)
proof−
from *xfHyp* **obtain** *u* **where** *xfuHyp*: (*u*,*x*,*f*) $\in$ *set* (*uInput* $\otimes$ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
show (*sol s[xfList←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* (λr. *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*:$\forall s\ g.\ \forall xf{\in}$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) $= \pi_2$ *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:$\forall$ *uxf* $\in$ *set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall t{\geq}0$. $\forall$ *xf* $\in$ *set xfList*. (($\lambda\tau$. (*sol s[xfList←uInput]* τ) (π_1 *xf*))
has-vderiv-on ($\lambda\tau$. (π_2 *xf*) (*sol s[xfList←uInput]* τ))) $\{0..t\}$
shows $\forall\ t \geq 0$. $\forall$ *xf* $\in$ *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*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
shows $\forall$ *x*. *x* $\notin$ (π_1(|*set xfList*|)) $\longrightarrow$ (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ *x*) = *0*
using *varsHyp* **apply**(*induct xfList uInput rule*: *list-induct2'*)
apply(*simp-all add*: *override-on-def varDiffs-def vdiff-def*)
by *clarsimp*

lemma *conds4InitState*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *xfHyp*:(*x*, *f*) $\in$ *set xfList*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *s x*
proof−
from *xfHyp* **obtain** *u* **where** *uxfHyp*:(*u*, *x*, *f*) $\in$ *set* (*uInput* $\otimes$ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
from *varsHyp* **have** *toZeroHyp*:(*sol s*) *x* = *s x* **using** *override-on-def xfHyp* **by** *auto*
from *uxfHyp* **and** *solHyp1* **have** *u 0* (*sol s*) = (*sol s*) *x* **by** *fastforce*
also have (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *u 0* (*sol s*)
using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*
ultimately show (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *s x* **using** *toZeroHyp* **by** *simp*
qed

lemma *conds4RestOfStrings*:
assumes *x* $\notin$ (π_1(|*set xfList*|)) $\cup$ *varDiffs*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = *s x*
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2'*)
by(*auto simp*: *varDiffs-def*)

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

theorem *dSolve-toSolve*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *s*.$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *s*.$\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*.
((λ*t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λ*t*. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0*..*t*}
and *uniqHyp*:$\forall$ *s*.$\forall$ *X*. *solvesStoreIVP X xfList s* $\longrightarrow$ ($\forall$ *t* $\geq$ *0*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = *X t*)
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolve*)

subgoal using *assms* **and** *conds4storeIVP-on-toSol* **by** *simp*
subgoal by (*simp add*: *uniqHyp*)
using *postCondHyp postCondHyp* **by** *simp*

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

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

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

lemma *ubcStoreUniqueSol*:
assumes *tHyp*:$t \geq 0$
assumes *contHyp*:$\forall$ *xf* $\in$ *set xfList*. *continuous-on* $(\{0..t\} \times UNIV)$
$(\lambda(t, (r::real)).\ (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ t))$
and *eqDerivs*:$\forall$ *xf* $\in$ *set xfList*. $\forall\ \tau \in \{0..t\}$. $(\pi_2\ xf)\ (\varphi_s\ \tau) = (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau)$
and *Fsolves*:*solvesStoreIVP* φ_s *xfList s*
and *solHyp*:*solvesStoreIVP* $(\lambda\ \tau.\ (sol\ s[xfList \leftarrow uInput]\ \tau))$ *xfList s*
shows $(sol\ s[xfList \leftarrow uInput]\ t) = \varphi_s\ t$
proof
fix x::*string* **show** $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x$
proof(*cases* $x \in (\pi_1 (\!|set\ xfList|\!)) \cup varDiffs$)
case *False*
then have *notInVars*:$x \notin (\pi_1 (\!|set\ xfList|\!)) \cup varDiffs$ **by** *simp*
from *solHyp* **have** $(sol\ s[xfList \leftarrow uInput]\ t)\ x = s\ x$
using *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
also from *Fsolves* **have** $\varphi_s\ t\ x = s\ x$ **using** *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
ultimately show $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x$ **by** *simp*
next case *True*
then have $x \in (\pi_1 (\!|set\ xfList|\!)) \vee x \in varDiffs$ **by** *simp*
from *this* **show** *?thesis*
proof
assume $x \in (\pi_1 (\!|set\ xfList|\!))$
from *this* **obtain** f **where** *xfHyp*:$(x, f) \in$ *set xfList* **by** *fastforce*

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

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

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

from *tHyp* **and** *contHyp* **have** $\forall$ *xf* $\in$ *set xfList*. *unique-on-bounded-closed 0* {*0..t*} (*s* (π_1 *xf*)) ($\lambda\tau$ *r*. (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ)) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*))
apply(*clarify*) **apply**(*rule conds4UniqSol*) **by** *auto*
from *this* **have** *3*:*unique-on-bounded-closed 0* {*0..t*} (*s x*) ($\lambda\tau$ *r*. *f* (*sol s*[*xfList*$\leftarrow$*uInput*] τ)) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*)) **using** *xfHyp* **by** *fastforce*
from *1 2* **and** *3* **show** (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = φ_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* = ∂ *y* **by** (*auto simp*: *varDiffs-def*)
show (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = φ_s *t x*
proof(*cases y* $\in$ *set* (*map* π_1 *xfList*))
case *True*
then obtain *f* **where** *xfHyp*:(*y*, *f*) $\in$ *set xfList* **by** *fastforce*
from *tHyp* **and** *Fsolves* **have** φ_s *t x* = *f* (φ_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*$\leftarrow$*uInput*] *t*) *x* = *0*
using *False solHyp tHyp solves-store-ivpD*(*2*) *xDef* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed

theorem *dSolveUBC*:
assumes *contHyp*:$\forall$ *s*. $\forall$ *t*$\geq$*0*. $\forall$ *xf* $\in$ *set xfList*. *continuous-on* ({*0..t*} $\times$ *UNIV*) (λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
and *solHyp*:$\forall$ *s*. *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*$\leftarrow$*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*$\leftarrow$*uInput*] *r*))
and *diffAssgn*: $\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *G* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) $\longrightarrow$ *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolve*)
prefer *2* **subgoal proof**(*clarify*)
fix *s*::*real store* **and** φ_s::*real* $\Rightarrow$ *real store* **and** *t*::*real*
assume *isSol*:*solvesStoreIVP* φ_s *xfList s* **and** *sHyp*:*0* $\leq$ *t*
from *this* **and** *uniqHyp* **have** $\forall$ *xf* $\in$ *set xfList*. $\forall$ *t* $\in$ {*0..t*}. (π_2 *xf*) (φ_s *t*) = (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) **by** *auto*
also have $\forall$ *xf* $\in$ *set xfList*. *continuous-on* ({*0..t*} $\times$ *UNIV*) (λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)) **using** *contHyp sHyp* **by** *blast*
ultimately show (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = φ_s *t*
using *sHyp isSol ubcStoreUniqueSol solHyp* **by** *simp*
qed using *assms* **by** *simp-all*

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

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

"Differential Invariant."

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

lemma *derivationLemma-baseCase*:
fixes *F*::*real* $\Rightarrow$ *real store*
assumes *solves*:*solvesStoreIVP F xfList a*
shows $\forall$ *x* $\in$ (*UNIV* $-$ *varDiffs*). $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in${*0..t*}.
((λ *t*. *F t x*) *has-vector-derivative F r* (∂ *x*)) (*at r within* {*0..t*})
proof
fix *x*
assume *x* $\in$ *UNIV* $-$ *varDiffs*
then have *notVarDiff*:$\forall$ *z*. *x* $\neq$ ∂ *z* **using** *varDiffs-def* **by** *fastforce*
show $\forall$ *t*$\geq$*0*. $\forall$ *r*$\in${*0..t*}. ((λt. *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*}. ((λ *r*. *a x*) *has-vector-derivative 0*) (*at r within* {*0..t*})
by (*auto intro*: *derivative-eq-intros*)
{**fix** *t r*::*real*

```
      assume t≥0 and r ∈ {0..t}
      hence ((λ s. a x) has-vector-derivative 0) (at r within {0..t}) by (simp add: constD)
      moreover have ⋀s. s ∈ {0..t} ⟹ (λ r. F r x) s = (λ r. a x) s
        using const by (simp add: ‹0 ≤ 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 ∈ {0..t}›)}
    hence isZero:∀ t≥0.∀ r∈{0..t}.((λ t. F t x)has-vector-derivative 0)(at r within {0..t})
      by blast
    from False solves and notVarDiff have ∀ t ≥ 0. F t (∂ 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 ≥ 0
    and termVarsHyp:∀ x ∈ trmVars η. x ∈ (UNIV − varDiffs)
  shows ∀ r∈{0..t}. ((λ s. ⟦η⟧_t (F s))has-vector-derivative ⟦∂_t η⟧_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 ∈ 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 η1 η2)
  then show ?case
    apply(clarsimp)
    by(rule derivative-intros, simp-all)
next
  case (Mult η1 η2)
  then show ?case
    apply(clarsimp)
    apply(subgoal-tac ((λs. ⟦η1⟧_t (F s) *_R ⟦η2⟧_t (F s)) has-vector-derivative
  ⟦∂_t η1⟧_t (F r) · ⟦η2⟧_t (F r) + ⟦η1⟧_t (F r) · ⟦∂_t η2⟧_t (F r)) (at r within {0..t}),simp)
    apply(rule-tac f'1=⟦∂_t η1⟧_t (F r) and g'1=⟦∂_t η2⟧_t (F r) in derivative-eq-intros(26))
    by (simp-all add: has-field-derivative-iff-has-vector-derivative)
qed

lemma diff-subst-prprty-4terms:
  assumes solves:∀ xf ∈ set xfList. F t (∂ (π_1 xf)) = π_2 xf (F t)
    and tHyp:(t::real) ≥ 0
    and listsHyp:map π_2 xfList = map tval uInput
    and termVarsHyp:trmVars η ⊆ (UNIV − varDiffs)
  shows ⟦∂_t η⟧_t (F t) = ⟦((map (vdiff ∘ π_1) xfList) ⊗ uInput)⟨∂_t η⟩⟧_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 × (real store ⇒ real)) list) and trmTail
  assume IH:⋀x. x ∉ varDiffs ⟹ map π_2 xfTail = map tval trmTail ⟹
∀ xf∈set xfTail. F t (∂ (π_1 xf)) = π_2 xf (F t) ⟹
F t (∂ x) = ⟦(map (vdiff ∘ π_1) xfTail ⊗ trmTail)⟨t_V (∂ x)⟩⟧_t (F t)
```

and $1{:}x \notin \mathit{varDiffs}$ **and** $2{:}\mathit{map}\ \pi_2\ ((y,\ g)\ \#\ \mathit{xfTail}) = \mathit{map}\ \mathit{tval}\ (\vartheta\ \#\ \mathit{trmTail})$
and $3{:}\forall \mathit{xf}{\in}\mathit{set}\ ((y,\ g)\ \#\ \mathit{xfTail}).\ F\ t\ (\partial\ (\pi_1\ \mathit{xf})) = \pi_2\ \mathit{xf}\ (F\ t)$
hence $*{:}[\![(\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfTail} \otimes \mathit{trmTail})\langle \mathit{Var}\ (\partial\ x)\rangle]\!]_t\ (F\ t) = F\ t\ (\partial\ x)$
using *tHyp* **by** *auto*
show $F\ t\ (\partial\ x) = [\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ ((y,\ g)\ \#\ \mathit{xfTail})) \otimes (\vartheta\ \#\ \mathit{trmTail}))\ \langle t_V\ (\partial\ x)\rangle]\!]_t\ (F\ t)$
proof(*cases* $x \in \mathit{set}\ (\mathit{map}\ \pi_1\ ((y,\ g)\ \#\ \mathit{xfTail}))$)
case *True*
then have $x = y \lor (x \neq y \land x \in \mathit{set}\ (\mathit{map}\ \pi_1\ \mathit{xfTail}))$ **by** *auto*
moreover
{assume $x = y$
from *this* **have** $((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ ((y,\ g)\ \#\ \mathit{xfTail})) \otimes (\vartheta\ \#\ \mathit{trmTail}))\langle t_V\ (\partial\ x)\rangle = \vartheta$
by *simp*
also from *3 tHyp* **have** $F\ t\ (\partial\ y) = g\ (F\ t)$
by *simp*
moreover from *2* **have** $[\![\vartheta]\!]_t\ (F\ t) = g\ (F\ t)$
by *simp*
ultimately have *?thesis*
by (*simp add*: ‹$x = y$›)**}**
moreover
{assume $x \neq y \land x \in \mathit{set}\ (\mathit{map}\ \pi_1\ \mathit{xfTail})$
then have $\partial\ x \neq \partial\ y$ **using** *vdiff-inj* **by** *auto*
from *this* **have** $((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ ((y,\ g)\ \#\ \mathit{xfTail})) \otimes (\vartheta\ \#\ \mathit{trmTail}))\ \langle t_V\ (\partial\ x)\rangle =$
$((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfTail}) \otimes \mathit{trmTail})\ \langle t_V\ (\partial\ x)\rangle$ **by** *simp*
hence *?thesis* **using** $*$ **by** *simp***}**
ultimately show *?thesis* **by** *blast*
next
case *False*
then have $((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ ((y,\ g)\ \#\ \mathit{xfTail})) \otimes (\vartheta\ \#\ \mathit{trmTail}))\ \langle t_V\ (\partial\ x)\rangle = t_V\ (\partial\ x)$
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*:$\mathit{trmVars}\ \eta \subseteq (\mathit{UNIV} - \mathit{varDiffs})$
and *initHyp*:$\forall x.\ x \notin \mathit{varDiffs} \longrightarrow b\ x = a\ x$
shows $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ b$
using *assms* **by**(*induction* η, *simp-all*)

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

lemma *dInvForTrms-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall \mathit{str}.\ \mathit{str} \notin (\pi_1(|\mathit{set}\ \mathit{xfList}|)) \longrightarrow st\ (\partial\ \mathit{str}) = 0) \longrightarrow$
$[\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\ \langle \partial_t\ \eta\rangle]\!]_t\ st = 0$
and *termVarsHyp*:$\mathit{trmVars}\ \eta \subseteq (\mathit{UNIV} - \mathit{varDiffs})$
and *listsHyp*:$\mathit{map}\ \pi_2\ \mathit{xfList} = \mathit{map}\ \mathit{tval}\ \mathit{uInput}$
shows $[\![\eta]\!]_t\ a = 0 \longrightarrow (\forall\ c.\ (a,c) \in (\mathit{ODEsystem}\ \mathit{xfList}\ \mathit{with}\ G) \longrightarrow [\![\eta]\!]_t\ c = 0)$
proof(*clarify*)
fix c **assume** *aHyp*:$[\![\eta]\!]_t\ a = 0$ **and** *cHyp*:$(a,\ c) \in \mathit{ODEsystem}\ \mathit{xfList}\ \mathit{with}\ G$
from *this* **obtain** $t{::}\mathit{real}$ **and** $F{::}\mathit{real} \Rightarrow \mathit{real}\ \mathit{store}$
where *tcHyp*:$t \geq 0 \land F\ t = c \land \mathit{solvesStoreIVP}\ F\ \mathit{xfList}\ a \land (\forall r{\in}\{0..t\}.\ G\ (F\ r))$
using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin \mathit{varDiffs} \longrightarrow F\ 0\ x = a\ x$
using *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$
using *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*

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

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

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

lemma *dInvForProps-prelim*:
assumes *substHyp*:
  $\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\![set\ xfList]\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ st \geq 0$
  and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
  and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a > 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c > 0)$
  and $[\![\eta]\!]_t\ a \geq 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c \geq 0)$
proof(*clarify*)

fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a > 0$ **and** *cHyp*:$(a,\ c) \in$ *ODEsystem xfList with G*
from *this* **obtain** *t*::*real* **and** *F*::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t \geq 0 \wedge F\ t = c \wedge$ *solvesStoreIVP F xfList a* $\wedge$ $(\forall r \in \{0..t\}.\ G\ (F\ r))$
using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin$ *varDiffs* $\longrightarrow F\ 0\ x = a\ x$
using *solves-store-ivpD(6)* **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$
using *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) > 0$
using *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative*
$[\![\partial_t\ \eta]\!]_t\ (F\ r))$ (*at r within* $\{0..t\}$)
using *derivationLemma termVarsHyp* **by** *blast*
have $(\forall t \geq 0.\ \forall\ xf \in$ *set xfList*. $F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using *tcHyp solves-store-ivpD(3)* **by** *blast*
hence $\forall r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r \in \{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD(2) tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*$:$\forall r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$
by *simp*
from *obs2* **and** *tcHyp* **have** $\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-derivative*
$(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))$ (*at r within* $\{0..t\}$)
by (*simp add*: *has-vector-derivative-def*)
hence $\exists r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![(\partial_t\ \eta)]\!]_t)\ (F\ r)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain *r* **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\wedge\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c > 0$
using *obs1 tcHyp* **by** (*smt mult-nonneg-nonneg*)
next
show $0 \leq [\![\eta]\!]_t\ a \longrightarrow (\forall c.\ (a,\ c) \in$ *ODEsystem xfList with G* $\longrightarrow 0 \leq [\![\eta]\!]_t\ c)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a \geq 0$ **and** *cHyp*:$(a,\ c) \in$ *ODEsystem xfList with G*
from *this* **obtain** *t*::*real* **and** *F*::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t \geq 0 \wedge F\ t = c \wedge$ *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*:$[\![\eta]\!]_t\ (F\ 0) \geq 0$
using *aHyp tcHyp* **by** *simp*
hence *obs2*:$\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative* $[\![\partial_t\ \eta]\!]_t\ (F\ r))$ (*at r within* $\{0..t\}$)
using *tcHyp derivationLemma termVarsHyp* **by** *blast*
have $(\forall t \geq 0.\ \forall\ xf \in$ *set xfList*. $F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using *tcHyp solves-store-ivpD(3)* **by** *blast*
hence $\forall r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ (F\ r)$
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r \in \{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle \partial_t\ \eta \rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD(2) tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*$:$\forall r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$
by *simp*
have $\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-derivative* $(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))$ (*at r within* $\{0..t\}$)
using *obs2 tcHyp* **by** (*simp add*: *has-vector-derivative-def*)
hence $\exists r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain *r* **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\wedge\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$

using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c \geq 0$
using *obs1 tcHyp* **by** (*smt mult-nonneg-nonneg*)
qed
qed

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

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

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

using *dInvForProps-prelim*(*1*) *listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) > 0$
using *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\eta]\!]_t\ (F\ t) > [\![\vartheta]\!]_t\ (F\ t)$
by *simp*
also have $[\![\vartheta \prec \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) < [\![\eta]\!]_t\ (F\ t))$
using *tcHyp* **by** *simp*
ultimately show *?case*
by *simp*
next
case (*Leq* $\vartheta\ \eta$)
hence $\forall\, st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$0 \le ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$
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 $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a \ge 0$
using *Leq.prems*(*1*) **by** *simp*
ultimately have $(\forall\, c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c \ge 0)$
using *dInvForProps-prelim*(*2*) *listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) \ge 0$
using *tcHyp cHyp* **by** *simp*
from *this* **have** $([\![\eta]\!]_t\ (F\ t) \ge [\![\vartheta]\!]_t\ (F\ t))$
by *simp*
also have $[\![\vartheta \preceq \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) \le [\![\eta]\!]_t\ (F\ t))$
using *tcHyp* **by** *simp*
ultimately show *?case*
by *simp*
next
case (*And* $\varphi 1\ \varphi 2$)
thus *?case*
by *simp*
next
case (*Or* $\varphi 1\ \varphi 2$)
thus *?case*
by *auto*
qed
qed

theorem *dInv*:
assumes $\forall\ st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\lceil\partial_P\ \varphi\rceil]\!]_P\ st$
and *termVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *phi-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 (*simp add*: *p2r-def*)
have $P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c)$
using *assms dInv-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c$
by *blast*
thus $(a,\ b) \in wp\ (ODEsystem\ xfList\ with\ G\ )\ \lceil P\rceil$
using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)
qed

theorem *dInvFinal*:

assumes $\forall\ st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\lceil\partial_P\ \varphi\rceil]\!]_P\ st$
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-f*:$F = [\![\varphi]\!]_P$
shows *PRE P (ODEsystem xfList with G) POST Q*
apply(*rule-tac* $C=[\![\varphi]\!]_P$ **in** *dCut*)
apply(*subgoal-tac* $\lceil F\rceil \subseteq wp\ (ODEsystem\ xfList\ with\ G)\ \lceil F\rceil$)
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

0.18.5 Rules Testing

In this section we test the recently developed rules with simple dynamical systems.

— Example of hybrid program verified with the rule dSolve and a single differential equation: $x' = v$.
lemma *motion-with-constant-velocity*:
$PRE\ (\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' > 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v''))]\ with\ (\lambda\ s.\ True))$
$POST\ (\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac* $uInput=[\lambda\ t\ s.\ s\ ''v'' \cdot t + s\ ''x'']$ **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal by** (*simp add*: *wp-rel vdiff-def add-strict-increasing2*)
apply (*simp-all add*: *vdiff-def varDiffs-def*)
prefer *2* **apply** (*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*)
apply (*clarify*, *rule-tac* $f'1=\lambda\ x.\ s\ ''v''$ **and** $g'1=\lambda\ x.\ 0$ **in** *derivative-intros*(*189*))
apply (*rule-tac* $f'1=\lambda\ x.0$ **and** $g'1=\lambda\ x.1$ **in** *derivative-intros*(*192*))
by (*auto intro*: *derivative-intros*)

Same hybrid program verified with dSolve and the system of ODEs: $x' = v, v' = a$. The uniqueness part of the proof requires a preliminary lemma.

lemma *flow-vel-is-galilean-vel*:
assumes *solHyp*:φ_s *solvesTheStoreIVP* $[(x, \lambda s.\ s\ v), (v, \lambda s.\ s\ a)]$ *withInitState s*
and *tHyp*:$r \le t$ **and** *rHyp*:$0 \le r$ **and** *distinct*:$x \ne v \wedge v \ne a \wedge x \ne a \wedge a \notin varDiffs$
shows $\varphi_s\ r\ v = s\ a \cdot r + s\ v$
proof−
from *assms* **have** *1*:$((\lambda t.\ \varphi_s\ t\ v)\ solves\text{-}ode\ (\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}\ UNIV \wedge \varphi_s\ 0\ v = s\ v$
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*:$((\lambda t.\ s\ a \cdot t + s\ v)\ solves\text{-}ode\ (\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}\ UNIV$
unfolding *solves-ode-def* **apply**(*subgoal-tac* $((\lambda x.\ s\ a \cdot x + s\ v)\ has\text{-}vderiv\text{-}on\ (\lambda x.\ s\ a))\ \{0..t\}$)
using *obs* **apply** (*simp add*: *has-vderiv-on-def*) **by**(*rule galilean-transform*)
have *3*:*unique-on-bounded-closed* $0\ \{0..t\}\ (s\ v)\ (\lambda t\ r.\ \varphi_s\ t\ a)\ UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t{+}1))$
apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*
apply(*rule continuous-intros*)
apply(*auto intro*: *continuous-intros*)

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

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

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

lemma *flow-vel-is-galilean-vel2*:
assumes *solHyp*:φ_s *solvesTheStoreIVP* [(*x*, λ*s*. *s v*), (*v*, λ*s*. − *s a*)] *withInitState s*
and *tHyp*:*r* $\leq$ *t* **and** *rHyp*:*0* $\leq$ *r* **and** *distinct*:*x* $\neq$ *v* $\wedge$ *v* $\neq$ *a* $\wedge$ *x* $\neq$ *a* $\wedge$ *a* $\notin$ *varDiffs*
shows φ_s *r v* = *s v* − *s a* · *r*
proof−
from *assms* **have** *1*:((λt. φ_s *t v*) *solves-ode* (λt *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*:((λt. − *s a* · *t* + *s v*) *solves-ode* (λt *r*. − φ_s *t a*)) {*0..t*} *UNIV*
unfolding *solves-ode-def* **apply**(*subgoal-tac* ((λx. − *s a* · *x* + *s v*) *has-vderiv-on* (λx. − *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*) (λt *r*. − φ_s *t a*) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*))
apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*
apply(*rule continuous-intros*)+
apply(*auto intro*: *continuous-intros*)
by (*metis continuous-on-const continuous-on-eq*)
thus φ_s *r v* = *s v* − *s a* · *r*
apply(*rule-tac unique-on-bounded-closed.unique-solution*[*of 0* {*0..t*} *s v*
(λt *r*. − φ_s *t a*) *UNIV* (*if t* = *0 then 1 else 1* / (*t* + *1*)) (λt. 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* $\geq$ *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* · *s* ''*v*'')) *ELSE* (''*v*'' ::= (λ *s*. *s* ''*v*''))))
POST (λ *s*. *0* $\leq$ *s* ''*x*'' $\wedge$ *s* ''*x*'' $\leq$ *H*)

apply(*simp*, *subst dS*[*of* [λ *t s*. $-$ *s* ''*g*'' $\cdot$ *t* ^ *2*/*2* + *s* ''*v*'' $\cdot$ *t* + *s* ''*x*'', λ *t s*. $-$ *s* ''*g*'' $\cdot$ *t* + *s* ''*v*'']])
— Given solution is actually a solution.
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton*, *safe*)
apply(*rule galilean-transform-eq*, *simp*)+
apply(*rule galilean-transform*)+
— Uniqueness of the flow.
apply(*rule ubcStoreUniqueSol*, *simp*)
apply(*simp add*: *vdiff-def del*: *comp-apply*)
apply(*auto intro*: *continuous-intros del*: *comp-apply*)[*1*]
apply(*rule continuous-intros*)+
apply(*simp add*: *vdiff-def*, *safe*)
apply(*clarsimp*) **subgoal for** *s X t* τ
apply(*rule flow-vel-is-galilean-vel2*[*of X* ''*x*''])
by(*simp-all add*: *varDiffs-def vdiff-def*)
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def*)
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton galilean-transform-eq galilean-transform*)
— Relation Between the guard and the postcondition.
by(*auto simp*: *vdiff-def p2r-def*)

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

lemma *system-where-the-guard-implies-the-postcondition2*:
PRE (λ *s*. *s* ''*x*'' = *0*)
(*ODEsystem* [(''*x*'',(λ *s*. *s* ''*x*'' + *1*))] *with* (λ *s*. *s* ''*x*'' $\geq$ *0*))
POST (λ *s*. *s* ''*x*'' $\geq$ *0*)
apply(*simp add*: *wp-rel*)
by (*auto simp*: *relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def*)

— Example of system proved with a differential invariant.
lemma *circular-motion*:
PRE (λ *s*. (*s* ''*x*'') $\cdot$ (*s* ''*x*'') + (*s* ''*y*'') $\cdot$ (*s* ''*y*'') $-$ (*s* ''*r*'') $\cdot$ (*s* ''*r*'') = *0*)
(*ODEsystem* [(''*x*'',(λ *s*. *s* ''*y*'')),(''*y*'',(λ *s*. $-$ *s* ''*x*''))] *with G*)
POST (λ *s*. (*s* ''*x*'') $\cdot$ (*s* ''*x*'') + (*s* ''*y*'') $\cdot$ (*s* ''*y*'') $-$ (*s* ''*r*'') $\cdot$ (*s* ''*r*'') = *0*)
apply(*rule-tac* η=(t_V ''*x*'')$\odot$(t_V ''*x*'') $\oplus$ (t_V ''*y*'')$\odot$(t_V ''*y*'') $\oplus$ ($\ominus$(t_V ''*r*'')$\odot$(t_V ''*r*''))
and *uInput*=[t_V ''*y*'', $\ominus$ (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.

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

— We revisit the two modes example from before, and prove it with invariants.
lemma *single-hop-ball-and-invariants*:
PRE $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0 \wedge 1 \geq c \wedge c \geq 0)$
$(((ODEsystem\ [(''x'',\ \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s.\ -\ s\ ''g'')]\ with\ (\lambda\ s.\ 0 \leq s\ ''x'')));$
$(IF\ (\lambda\ s.\ s\ ''x'' = 0)\ THEN\ (''v'' ::= (\lambda\ s.\ -\ c \cdot s\ ''v''))\ ELSE\ (''v'' ::= (\lambda\ s.\ s\ ''v''))))$
POST $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*simp*, *rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \prec (t_V\ ''g'')$ **and** *uInput*=$[t_V\ ''v'',\ \ominus\ t_V\ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x=''g''$ **in** *allE*, *simp*)
apply(*rule-tac* $C =\lambda\ s.\ \ s\ ''v'' \leq 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ ''v'') \preceq (t_C\ 0)$ **and** *uInput*=$[t_V\ ''v'',\ \ominus\ t_V\ ''g'']$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' \leq\ \ H$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ ''x'') \preceq (t_C\ H)$ **and** *uInput*=$[t_V\ ''v'',\ \ominus\ t_V\ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *varDiffs-def vdiff-def*)
using *dWeakening* **by** *simp*

— Finally, we add a well known example in the hybrid systems community, the bouncing ball.
lemma *bouncing-ball-invariant*:$0 \leq x \Longrightarrow 0 < g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \Longrightarrow (x{::}real) \leq H$
proof−
assume $0 \leq x$ **and** $0 < g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v$
then have $v \cdot v = 2 \cdot g \cdot H - 2 \cdot g \cdot x \wedge 0 < g$ **by** *auto*
hence $*{:}v \cdot v = 2 \cdot g \cdot (H - x) \wedge 0 < g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
from *this* **have** $(v \cdot v)/(2 \cdot g) = (H - x)$ **by** *auto*
also from $*$ **have** $(v \cdot v)/(2 \cdot g) \geq 0$
using *divide-nonneg-pos* **by** *fastforce*
ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
PRE $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0)$
$((ODEsystem\ [(''x'',\ \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s.\ -\ s\ ''g'')]\ with\ (\lambda\ s.\ 0 \leq s\ ''x''));$
$(IF\ (\lambda\ s.\ s\ ''x'' = 0)\ THEN\ (''v'' ::= (\lambda\ s.\ -\ s\ ''v''))\ ELSE\ Id))^*$
POST $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*rule wp-loopI*[*of* - $\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge$
$2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - (s\ ''v'' \cdot s\ ''v'')$])
apply(*simp*, *clarsimp simp*: *bouncing-ball-invariant*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = ((t_C\ 0) \prec (t_V\ ''g''))$ **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*)
by (*rule dWeakening*, *clarsimp*)

end