

Verification Components for Hybrid Systems

Jonathan Julián Huerta y Munive

December 14, 2021

Abstract

These components formalise a semantic framework for the deductive verification of hybrid systems. They support reasoning about continuous evolutions of hybrid programs in the style of differential dynamic logic. Vector fields or flows model these evolutions, and their verification is done with invariants for the former or orbits for the latter. Laws of modal Kleene algebra or categorical predicate transformers implement the verification condition generation. Examples show the approach at work.

Contents

1	Introductory Remarks	3
2	Hybrid Systems Preliminaries	4
2.1	Real numbers	4
2.2	Single variable derivatives	6
2.3	Intermediate Value Theorem	8
2.4	Filters	9
2.5	Multivariable derivatives	10
3	Ordinary Differential Equations	12
3.1	Initial value problems and orbits	13
3.2	Differential Invariants	14
3.3	Picard-Lindelöf	19
3.4	Flows for ODEs	23
4	Verification components for hybrid systems	28
4.1	Verification of regular programs	28
4.2	Verification of hybrid programs	32
4.3	Derivation of the rules of dL	34
4.4	Examples	37
4.4.1	Pendulum	37
4.4.2	Bouncing Ball	38

4.4.3	Thermostat	40
4.4.4	Tank	43
5	Verification components with Predicate Transformers	44
5.1	Verification of regular programs	45
5.2	Verification of hybrid programs	48
5.3	Derivation of the rules of dL	50
5.4	Examples	53
5.4.1	Pendulum	53
5.4.2	Bouncing Ball	54
5.4.3	Thermostat	56
5.4.4	Tank	59
6	Verification components with MKA	60
6.1	Verification in AKA	61
6.2	Relational model	64
6.3	Verification of hybrid programs	66
6.3.1	Regular programs	66
6.3.2	Evolution commands	68
6.3.3	Derivation of the rules of dL	70
6.4	Examples	74
6.4.1	Pendulum	74
6.4.2	Bouncing Ball	75
6.4.3	Thermostat	77
6.4.4	Tank	80
6.5	State transformer model	81
6.6	Verification of hybrid programs	83
6.6.1	Regular programs	83
6.6.2	Evolution commands	85
6.6.3	Derivation of the rules of dL	88
6.7	Examples	91
6.7.1	Pendulum	91
6.7.2	Bouncing Ball	92
6.7.3	Thermostat	95
6.7.4	Tank	97
7	Verification components with KAT	98
7.1	Hoare logic in KAT	99
7.2	refinement KAT	101
7.3	Verification of hybrid programs	104
7.3.1	Regular programs	105
7.3.2	Evolution commands	108
7.4	Refinement Components	110
7.5	Derivation of the rules of dL	115

7.6	Examples	116
7.6.1	Pendulum	117
7.6.2	Bouncing Ball	118
7.6.3	Thermostat	121
7.6.4	Water tank	125
7.6.5	Tank	125
7.7	Verification of hybrid programs	130
7.7.1	Regular programs	130
7.7.2	Evolution commands	134
7.8	Refinement Components	136
7.9	Derivation of the rules of dL	141
7.10	Examples	142
7.10.1	Pendulum	143
7.10.2	Bouncing Ball	144
7.10.3	Thermostat	147
7.10.4	Water tank	151
7.10.5	Tank	151

1 Introductory Remarks

These theories implement verification components for hybrid programs in the style of differential dynamic logic [6], an approach for deductive verification of hybrid systems. Following [4], we use modal Kleene algebra, which subsumes the propositional part of dynamic logic, to automatically derive verification conditions for the program flow. Alternatively we also use categorical predicate transformers as formalised in [7]. These conditions 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) as in [5]. 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. In all versions of our verification components we also derive domain specific rules of differential dynamic logic and prove a correctness specification of three hybrid systems using each of our procedures for reasoning with continuous evolutions. 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 [6, 2], the general shallow embedding approach for building verification components with Isabelle can be found in [1]. For more

details on modal Kleene algebra see [3]; a paper with a detailed overview of the verification components in this entry and the mathematical concepts employed to build them will be available soon on ArXiv.

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-Lindelof-Qualitative*
begin

— Notation

bundle *derivative-notation*

begin

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)

end

bundle *derivative-no-notation*

begin

notation *has-vderiv-on* (**infix** (*has'-vderiv'-on*) 50)

no-notation *has-derivative* ((1(*D* - \mapsto (-))/ -) [65,65] 61)

and *has-vderiv-on* ((1(*D* - = (-)/ on -) [65,65] 61)

end

unbundle *derivative-notation*

notation *norm* ($\|-\|$)

2.1 Real numbers

lemma *abs-le-eq*:

shows $(r::real) > 0 \implies (|x| < r) = (-r < x \wedge x < r)$

and $(r::real) > 0 \implies (|x| \leq r) = (-r \leq x \wedge x \leq r)$

by *linarith+*

lemma *real-ivl-eqs*:

assumes $0 < r$

shows $ball\ x\ r = \{x-r < \dots < x+r\}$ **and** $\{x-r < \dots < x+r\} = \{x-r < \dots < x+r\}$
and $ball\ (r / 2)\ (r / 2) = \{0 < \dots < r\}$ **and** $\{0 < \dots < r\} = \{0 < \dots < r\}$
and $ball\ 0\ r = \{-r < \dots < r\}$ **and** $\{-r < \dots < r\} = \{-r < \dots < r\}$
and $cball\ x\ r = \{x-r \dots x+r\}$ **and** $\{x-r \dots x+r\} = \{x-r \dots x+r\}$
and $cball\ (r / 2)\ (r / 2) = \{0 \dots r\}$ **and** $\{0 \dots r\} = \{0 \dots r\}$
and $cball\ 0\ r = \{-r \dots r\}$ **and** $\{-r \dots r\} = \{-r \dots r\}$
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

lemma *sum-eq-Sum:*
assumes *inj-on f A*
shows $(\sum_{x \in A}. f\ x) = (\sum \{f\ x \mid x. x \in A\})$
proof –
have $(\sum \{f\ x \mid x. x \in A\}) = (\sum (f\ ` A))$
apply (*auto simp: image-def*)
by (*rule-tac f=Sum in arg-cong, auto*)
also have $\dots = (\sum_{x \in A}. f\ x)$
by (*subst sum.image-eq[OF assms], simp*)
finally show $(\sum_{x \in A}. f\ x) = (\sum \{f\ x \mid x. x \in A\})$
by *simp*
qed

lemma *triangle-norm-vec-le-sum: $\|x\| \leq (\sum_{i \in UNIV}. \|x\ \$\ i\|)$*
by (*simp add: L2-set-le-sum norm-vec-def*)

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 $g \text{ ' } T$
and $D g = g'$ on T
and $h = (\lambda t. g' t *_{\mathbb{R}} f' (g t))$
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 $T \implies g = (\lambda t. - f' t) \implies D (\lambda t. - f t) = g$ on T
using *has-vderiv-on-uminus* **by** *auto*

lemma *vderiv-npowI* [*poly-derivatives*]:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $n \geq 1$ **and** $D f = f'$ on T **and** $g = (\lambda t. n * (f' t) * (f t)^{\wedge}(n-1))$
shows $D (\lambda t. (f t)^{\wedge}n) = g$ 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 $\forall t \in T. g t \neq (0 :: \text{real})$ **and** $D f = f'$ on T **and** $D g = g'$ on T
and $h = (\lambda t. (f' t * g t - f t * (g' t)) / (g t)^{\wedge}2)$
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** $f'=\lambda t. - 1/t^{\wedge}2$, *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 $D (f :: \text{real} \Rightarrow \text{real}) = f'$ 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 $T \Longrightarrow g = (\lambda t. - a * f' t / (f t)^2)$
 $\Longrightarrow D (\lambda t. a/(f t)) = g$ 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 $c \neq 0 \Longrightarrow D (\lambda t. a5 * t^5 + a3 * (t^3 / c) - a2 * \exp (t^2) + a1 * \cos t + a0) =$
 $(\lambda t. 5 * a5 * t^4 + 3 * a3 * (t^2 / c) - 2 * a2 * t * \exp (t^2) - a1 * \sin t)$
on T
by(*auto intro!*: *poly-derivatives simp: power2-eq-square*)

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

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

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** $a \leq x \leq b$ **and** $?h x = 0$
using *IVT'[of ?h] <a ≤ b>* **by** *blast*
thus *?thesis*
using $\langle a \leq b \rangle$ **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 ≤ 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 $\langle a \geq b \rangle$ **unfolding** *closed-segment-eq-real-ivl* **by** *auto force*
qed

2.4 Filters

lemma *eventually-at-within-mono*:

assumes $t \in \text{interior } T$ **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 $t::'a::\{\text{perfect-space}, \text{t2-space}\}$

assumes $t \in \text{interior } T$ **and** $T \subseteq S$

shows *netlimit* (at t within S) = t

using *assms*(1) *interior-mono*[*OF* $\langle T \subseteq S \rangle$] *netlimit-within-interior* **by** *auto*

lemma *has-derivative-at-within-mono*:

assumes $(t::\text{real}) \in \text{interior } T$ **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 $P :: ('a::\text{finite}) \Rightarrow 'b \Rightarrow \text{bool}$

assumes $h:\forall i. \text{eventually } (P i) F$

shows *eventually* $(\lambda x. \forall i. P i x) F$

proof(*unfold* *eventually-def*)

let $?F = \text{Rep-filter } F$

have $\text{obs}:\forall i. ?F (P i)$

using h **by** *auto*

have $?F (\lambda x. \forall i \in \text{UNIV}. P i x)$

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 $P :: ('a::\text{finite}) \Rightarrow 'b \Rightarrow \text{bool}$

assumes $h1:\forall i. \text{eventually } (P i) F$

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

2.5 Multivariable derivatives

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*)

lemma *has-derivative-vec-nth*[*derivative-intros*]:
 $D (\lambda s. s \$ i) \mapsto (\lambda s. s \$ i)$ (at x within S)
by (*clarsimp simp: has-derivative-within*) *standard*

lemma *bounded-linear-component*:
(*bounded-linear f*) \longleftrightarrow ($\forall i. \text{bounded-linear } (\lambda x. (f x) \$ i)$) (**is** ?lhs = ?rhs)

proof

assume ?lhs

thus ?rhs

apply(*clarsimp, rule-tac f=(λx. x \$ i) in bounded-linear-compose*)

apply(*simp-all add: bounded-linear-def bounded-linear-axioms-def linear-iff*)

by (*rule-tac x=1 in exI, clarsimp*) (*meson Finite-Cartesian-Product.norm-nth-le*)

next

assume ?rhs

hence ($\forall i. \exists K. \forall x. \|f x \$ i\| \leq \|x\| * K$) *linear f*

by (*auto simp: bounded-linear-def bounded-linear-axioms-def linear-iff vec-eq-iff*)

then obtain F **where** $F: \bigwedge i x. \|f x \$ i\| \leq \|x\| * F i$

by *metis*

have $\|f x\| \leq \|x\| * \text{sum } F \text{ UNIV}$ **for** x

proof –

have $\text{norm } (f x) \leq (\sum i \in \text{UNIV}. \|f x \$ i\|)$

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

also have $\dots \leq (\sum i \in \text{UNIV}. \text{norm } x * F i)$

by (*metis F sum-mono*)

also have $\dots = \text{norm } x * \text{sum } F \text{ UNIV}$

by (*simp add: sum-distrib-left*)

finally show ?thesis .

qed

then show ?lhs

by (*force simp: bounded-linear-def bounded-linear-axioms-def <linear f>*)

qed

lemma *open-preimage-nth*:
 $\text{open } S \implies \text{open } \{s :: ('a :: \text{real-normed-vector } ^\sim n :: \text{finite}). s \$ i \in S\}$
unfolding *open-contains-ball* **apply** *clarsimp*
apply(*erule-tac x=x\$i in ballE; clarsimp*)

apply(*rule-tac* $x=e$ **in** exI ; *clarsimp simp: dist-norm subset-eq ball-def*)
apply(*rename-tac* $x e y$, *erule-tac* $x=y\$i$ **in** $allE$)
using *Finite-Cartesian-Product.norm-nth-le*
by (*metis le-less-trans vector-minus-component*)

lemma *tendsto-nth-iff*: — following $(?f \longrightarrow ?l) ?F = (\forall i \in \text{Basis}. ((\lambda x. ?f x \cdot i) \longrightarrow ?l \cdot i) ?F)$

fixes $l::'a::\text{real-normed-vector}^{\sim}n::\text{finite}$

defines $m \equiv \text{real CARD}('n)$

shows $(f \longrightarrow l) F \longleftrightarrow (\forall i. ((\lambda x. f x \$ i) \longrightarrow l \$ i) F)$ (**is** $?lhs = ?rhs$)

proof

assume $?lhs$

thus $?rhs$

unfolding *tendsto-def*

by (*clarify*, *drule-tac* $x=\{s. s \$ i \in S\}$ **in** *spec*) (*auto simp: open-preimage-nth*)

next

assume $?rhs$

thus $?lhs$

proof(*unfold tendsto-iff dist-norm, clarify*)

fix $\varepsilon::\text{real}$ **assume** $0 < \varepsilon$

assume *evnt-h*: $\forall i \varepsilon. 0 < \varepsilon \longrightarrow (\forall_F x \text{ in } F. \|f x \$ i - l \$ i\| < \varepsilon)$

{fix x **assume** *hyp*: $\forall i. \|f x \$ i - l \$ i\| < (\varepsilon/m)$

have $\|f x - l\| \leq (\sum_{i \in \text{UNIV}. \|f x \$ i - l \$ i\|})$

using *triangle-norm-vec-le-sum[of f x - l]* **by** *auto*

also have $\dots < (\sum_{(i::'n) \in \text{UNIV}. (\varepsilon/m))}$

apply(*rule sum-strict-mono[of UNIV* $\lambda i. \|f x \$ i - l \$ i\| \lambda i. \varepsilon/m$])

using *hyp* **by** *auto*

also have $\dots = m * (\varepsilon/m)$

unfolding *assms* **by** *simp*

finally have $\|f x - l\| < \varepsilon$

unfolding *assms* **by** *simp*}

hence *key*: $\bigwedge x. \forall i. \|f x \$ i - l \$ i\| < (\varepsilon/m) \implies \|f x - l\| < \varepsilon$

by *blast*

have *obs*: $\forall_F x \text{ in } F. \forall i. \|f x \$ i - l \$ i\| < (\varepsilon/m)$

apply(*rule eventually-all-finite*)

using $\langle 0 < \varepsilon \rangle$ *evnt-h* **unfolding** *assms* **by** *auto*

thus $\forall_F x \text{ in } F. \|f x - l\| < \varepsilon$

by (*rule eventually-mono[OF - key]*, *simp*)

qed

qed

lemma *has-derivative-component[simp]*: — following $D ?f \mapsto ?f'$ at $?a$ within $?S$
 $= (\forall i \in \text{Basis}. D (\lambda x. ?f x \cdot i) \mapsto (\lambda x. ?f' x \cdot i) \text{ at } ?a \text{ within } ?S)$

$(D f \mapsto f' \text{ at } x \text{ within } S) \longleftrightarrow (\forall i. D (\lambda s. f s \$ i) \mapsto (\lambda s. f' s \$ i) \text{ at } x \text{ within } S)$

by (*simp add: has-derivative-within tendsto-nth-iff*

bounded-linear-component all-conj-distrib)

lemma *has-vderiv-on-component[simp]*:

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

shows $(D x = x' \text{ on } T) = (\forall i. D (\lambda t. x t \$ i) = (\lambda t. x' t \$ i) \text{ on } T)$
unfolding *has-vderiv-on-def has-vector-derivative-def by auto*

lemma *frechet-tendsto-vec-lambda:*

fixes $f::\text{real} \Rightarrow ('a::\text{banach}) \wedge ('m::\text{finite})$ **and** $x::\text{real}$ **and** $T::\text{real set}$
defines $x_0 \equiv \text{netlimit (at } x \text{ within } T)$ **and** $m \equiv \text{real CARD('m)}$
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)
using *assms by (simp add: tendsto-nth-iff)*

lemma *tendsto-norm-bound:*

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

lemma *tendsto-zero-norm-bound:*

$\forall x. \|G x\| \leq \|F x\| \implies (F \longrightarrow 0) \text{ net} \implies (G \longrightarrow 0) \text{ net}$
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-tendsto-vec-nth:*

fixes $f::\text{real} \Rightarrow ('a::\text{real-normed-vector}) \wedge m$
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*

end

3 Ordinary Differential Equations

Vector fields $f::\text{real} \Rightarrow 'a \Rightarrow ('a::\text{real-normed-vector})$ represent systems of ordinary differential equations (ODEs). Picard-Lindelof's theorem guarantees existence and uniqueness of local solutions to initial value problems involving Lipschitz continuous vector fields. A (local) flow $\varphi::\text{real} \Rightarrow 'a \Rightarrow ('a::\text{real-normed-vector})$ for such a system is the function that maps initial conditions to their unique solutions. In dynamical systems, the set of all points $\varphi t s::'a$ for a fixed $s::'a$ is the flow's orbit. If the orbit of each $s \in$

I is contained 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
```

3.1 Initial value problems and orbits

```
notation image (P)
```

```
lemma image-le-pred[simp]: (P f A ⊆ {s. G s}) = (∀ x ∈ A. G (f x))
  unfolding image-def by force
```

```
definition ivp-sols :: (real ⇒ 'a ⇒ ('a::real-normed-vector)) ⇒ ('a ⇒ real set) ⇒
'a set ⇒
  real ⇒ 'a ⇒ (real ⇒ 'a) set (Sols)
  where Sols f U S t₀ s = {X ∈ U s → S. (D X = (λt. f t (X t)) on U s) ∧ X t₀
= s ∧ t₀ ∈ U s}
```

```
lemma ivp-solsI:
  assumes D X = (λt. f t (X t)) on U s and X t₀ = s
    and X ∈ U s → S and t₀ ∈ U s
  shows X ∈ Sols f U S t₀ s
  using assms unfolding ivp-sols-def by blast
```

```
lemma ivp-solsD:
  assumes X ∈ Sols f U S t₀ s
  shows D X = (λt. f t (X t)) on U s and X t₀ = s
    and X ∈ U s → S and t₀ ∈ U s
  using assms unfolding ivp-sols-def by auto
```

```
lemma in-ivp-sols-subset:
  t₀ ∈ (U s) ⇒ (U s) ⊆ (T s) ⇒ X ∈ Sols f T S t₀ s ⇒ X ∈ Sols f U S t₀ 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 ≡ {τ ∈ U. τ ≤ t}
```

```
definition g-orbit :: (('a::ord) ⇒ 'b) ⇒ ('b ⇒ bool) ⇒ 'a set ⇒ 'b set (γ)
  where γ X G U = ⋃{P X (down U t) | t. P X (down U t) ⊆ {s. G s}}
```

```
lemma g-orbit-eq:
  fixes X::('a::preorder) ⇒ 'b
  shows γ X G U = {X t | t. t ∈ U ∧ (∀ τ ∈ down U t. G (X τ))}
  unfolding g-orbit-def using order-trans by auto blast
```

```
definition g-orbital :: (real ⇒ 'a ⇒ 'a) ⇒ ('a ⇒ bool) ⇒ ('a ⇒ real set) ⇒ 'a set
```

\Rightarrow *real* \Rightarrow
 ('a::real-normed-vector) \Rightarrow 'a set
 where *g-orbital* f G U S t₀ s = $\bigcup \{ \gamma X G (U s) \mid X. X \in \text{ivp-sols } f U S t_0 s \}$

lemma *g-orbital-eq*: *g-orbital* f G U S t₀ s =
 $\{ X t \mid t X. t \in U s \wedge \mathcal{P} X (\text{down } (U s) t) \subseteq \{s. G s\} \wedge X \in \text{Sols } f U S t_0 s \}$
unfolding *g-orbital-def ivp-sols-def g-orbit-eq* **by** *auto*

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

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

lemma *g-orbital* f G U S t₀ s = $\{ X t \mid t X. X t \in \gamma X G (U s) \wedge X \in \text{Sols } f U S t_0 s \}$
unfolding *g-orbital-eq g-orbit-eq* **by** *auto*

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

lemma *g-orbital* f G U S t₀ s \subseteq *g-orbital* f ($\lambda s. \text{True}$) U S t₀ s
unfolding *g-orbital-eq* **by** *auto*

no-notation *g-orbit* (γ)

3.2 Differential Invariants

definition *diff-invariant* :: ('a \Rightarrow bool) \Rightarrow (real \Rightarrow ('a::real-normed-vector) \Rightarrow 'a)
 \Rightarrow
 ('a \Rightarrow real set) \Rightarrow 'a set \Rightarrow real \Rightarrow ('a \Rightarrow bool) \Rightarrow bool
 where *diff-invariant* I f U S t₀ G \equiv $(\bigcup \circ (\mathcal{P} (\text{g-orbital } f G U S t_0))) \{s. I s\} \subseteq \{s. I s\}$

lemma *diff-invariant-eq*: *diff-invariant* I f U S t₀ G =
 $(\forall s. I s \longrightarrow (\forall X \in \text{Sols } f U S t_0 s. (\forall t \in U s. (\forall \tau \in (\text{down } (U s) t). G (X \tau)) \longrightarrow I (X t))))$
unfolding *diff-invariant-def g-orbital-eq image-le-pred* **by** *auto*

lemma *diff-inv-eq-inv-set*:
diff-invariant I f U S t₀ G = $(\forall s. I s \longrightarrow (\text{g-orbital } f G U S t_0 s) \subseteq \{s. I s\})$
unfolding *diff-invariant-eq g-orbital-eq image-le-pred* **by** *auto*

lemma *diff-invariant I f U S t₀ (λs. True) ⇒ diff-invariant I f U S t₀ 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: $\bigwedge s. s \in S \implies \text{is-interval } (U s)$*

and *dX: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies (D (\lambda\tau. \mu(X \tau) - \nu(X \tau))) = ((*_R) 0) \text{ on } U(X t_0))$*

shows *diff-invariant (λs. μ s = ν s) f U S t₀ G*

proof(*simp add: diff-invariant-eq ivp-sols-def, clarsimp*)

fix *X t*

assume *xivp: $D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U (X t_0) \mu (X t_0) = \nu (X t_0) X \in U (X t_0) \rightarrow S$*

and *tHyp: $t \in U (X t_0)$ and t0Hyp: $t_0 \in U (X t_0)$*

hence *$\{t_0 \dashv\vdash t\} \subseteq U (X t_0)$*

using *closed-segment-subset-interval[OF Uhyp t0Hyp tHyp] by blast*

hence *$D (\lambda\tau. \mu (X \tau) - \nu (X \tau)) = (\lambda\tau. \tau *_R 0) \text{ on } \{t_0 \dashv\vdash t\}$*

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 ⇒ real*

assumes *Uhyp: $\bigwedge s. s \in S \implies \text{is-interval } (U s)$*

and *Gg: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies (\forall \tau \in U(X t_0). \tau > t_0 \longrightarrow G (X \tau) \longrightarrow \mu' (X \tau) \geq \nu' (X \tau))$*

and *Gl: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies (\forall \tau \in U(X t_0). \tau < t_0 \longrightarrow \mu' (X \tau) \leq \nu' (X \tau))$*

and *dX: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies D (\lambda\tau. \mu(X \tau) - \nu(X \tau)) = (\lambda\tau. \mu'(X \tau) - \nu'(X \tau)) \text{ on } U(X t_0)$*

shows *diff-invariant (λs. ν s ≤ μ s) f U S t₀ 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: $D X = (\lambda x. f x (X x)) \text{ on } U (X t_0) \nu (X t_0) \leq \mu (X t_0) X \in U (X t_0) \rightarrow S$*

assume *tHyp: $t \in U (X t_0)$ and t0Hyp: $t_0 \in U (X t_0)$*

hence *obs1: $\{t_0 \dashv\vdash t\} \subseteq U (X t_0) \{t_0 < \dashv\vdash < t\} \subseteq U (X t_0)$*

using *closed-segment-subset-interval[OF Uhyp t0Hyp tHyp] xivp(3) segment-open-subset-closed*

by (*force,metis PiE <X t₀ ∈ S ⇒ {t₀ ⋯ t} ⊆ U (X t₀)> dual-order.trans*)

hence *obs2: $D (\lambda\tau. \mu (X \tau) - \nu (X \tau)) = (\lambda\tau. \mu' (X \tau) - \nu' (X \tau)) \text{ on } \{t_0 \dashv\vdash t\}$*

using *has-vderiv-on-subset[OF dX[OF xivp(1)]] by auto*

{assume *t ≠ t₀*

then obtain *r where rHyp: $r \in \{t_0 < \dashv\vdash < t\}$*

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*: $\bigwedge \tau. \tau \in U(X t_0) \implies \tau > t_0 \implies G(X \tau) \implies \mu'(X \tau) \geq \nu'$
 $(X \tau)$
 $\bigwedge \tau. \tau \in U(X t_0) \implies \tau < t_0 \implies \mu'(X \tau) \leq \nu'(X \tau)$
using *Gg[OF xivp(1)] Gl[OF xivp(1)]* **by** *auto*
have $t > t_0 \implies r > t_0 \wedge G(X r) \neg t_0 \leq t \implies r < t_0 \wedge r \in U(X t_0)$
using $\langle r \in \{t_0 < \dots < t\} \rangle$ *obs1 Ghyp*
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl* **by** *auto*
moreover **have** $r > t_0 \implies G(X r) \implies (\mu'(X r) - \nu'(X r)) \geq 0 \wedge r < t_0 \implies$
 $(\mu'(X r) - \nu'(X r)) \leq 0$
using *primed(1,2)[OF \langle r \in U(X t_0) \rangle]* **by** *auto*
ultimately **have** $(t - t_0) * (\mu'(X r) - \nu'(X r)) \geq 0$
by *(case-tac t \ge 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 $\mu::'a::\text{banach} \Rightarrow \text{real}$
assumes *Uhyp*: $\bigwedge s. s \in S \implies \text{is-interval}(U s)$
and *Gg*: $\bigwedge X. (D X = (\lambda \tau. f \tau(X \tau)) \text{ on } U(X t_0)) \implies (\forall \tau \in U(X t_0). \tau > t_0 \implies G(X \tau) \implies \mu'(X \tau) \geq \nu'(X \tau))$
and *Gl*: $\bigwedge X. (D X = (\lambda \tau. f \tau(X \tau)) \text{ on } U(X t_0)) \implies (\forall \tau \in U(X t_0). \tau < t_0 \implies \mu'(X \tau) \leq \nu'(X \tau))$
and *dX*: $\bigwedge X. (D X = (\lambda \tau. f \tau(X \tau)) \text{ on } U(X t_0)) \implies D(\lambda \tau. \mu(X \tau) - \nu(X \tau)) = (\lambda \tau. \mu'(X \tau) - \nu'(X \tau)) \text{ on } U(X t_0)$
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*: $\forall \tau. \tau \in U(X t_0) \wedge \tau \leq t \implies G(X \tau)$
assume *xivp*: $D X = (\lambda x. f x(X x)) \text{ on } U(X t_0) \wedge \nu(X t_0) < \mu(X t_0) \wedge X \in U(X t_0) \rightarrow S$
assume *tHyp*: $t \in U(X t_0)$ **and** *t0Hyp*: $t_0 \in U(X t_0)$
hence *obs1*: $\{t_0 \dots t\} \subseteq U(X t_0) \wedge \{t_0 < \dots < t\} \subseteq U(X t_0)$
using *closed-segment-subset-interval[OF Uhyp t0Hyp tHyp] xivp(3) segment-open-subset-closed*
by *(force,metis PiE \langle X t_0 \in S \implies \{t_0 \dots t\} \subseteq U(X t_0) \rangle dual-order.trans)*
hence *obs2*: $D(\lambda \tau. \mu(X \tau) - \nu(X \tau)) = (\lambda \tau. \mu'(X \tau) - \nu'(X \tau)) \text{ on } \{t_0 \dots t\}$
using *has-vderiv-on-subset[OF dX[OF xivp(1)]]* **by** *auto*
{assume $t \neq t_0$
then **obtain** r **where** *rHyp*: $r \in \{t_0 < \dots < t\}$
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 *mut*: $\mu(X t) - \nu(X t) = (t - t_0) * (\mu'(X r) - \nu'(X r)) + (\mu(X t_0) - \nu(X t_0))$
by force
have primed: $\bigwedge \tau. \tau \in U (X t_0) \implies \tau > t_0 \implies G (X \tau) \implies \mu' (X \tau) \geq \nu' (X \tau)$
 $(X \tau)$
 $\bigwedge \tau. \tau \in U (X t_0) \implies \tau < t_0 \implies \mu' (X \tau) \leq \nu' (X \tau)$
using *Gg[OF xivp(1)] Gl[OF xivp(1)] by auto*
have $t > t_0 \implies r > t_0 \wedge G (X r) \neg t_0 \leq t \implies r < t_0 \wedge r \in U (X t_0)$
using $\langle r \in \{t_0 < \dots < t\} \rangle$ *obs1 Ghyp*
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl by auto*
moreover have $r > t_0 \implies G (X r) \implies (\mu'(X r) - \nu'(X r)) \geq 0 \wedge r < t_0 \implies$
 $(\mu'(X r) - \nu'(X r)) \leq 0$
using *primed(1,2)[OF \langle r \in U (X t_0) \rangle] by auto*
ultimately have $(t - t_0) * (\mu'(X r) - \nu'(X r)) \geq 0$
by (*case-tac t ≥ t₀, 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 *mut by simp*
thus $\nu (X t) < \mu (X t)$
using *xivp by blast*
qed

lemma *diff-invariant-nleq-rule*:
fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
shows *diff-invariant* $(\lambda s. \neg \nu s \leq \mu s) f U S t_0 G \longleftrightarrow \text{diff-invariant} (\lambda s. \nu s >$
 $\mu s) f U S t_0 G$
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 $\mu::'a::\text{banach} \Rightarrow \text{real}$
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** $X::\text{real} \Rightarrow 'a$ **and** $t::\text{real}$
assume $\nu s \neq \mu s$ **and** *Xhyp*: $X \in \text{Sols } f U S t_0 s$
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 $\nu s < \mu s \implies \nu (X t) < \mu (X t)$
using *assms(1) Xhyp thyp Ghyp unfolding diff-invariant-eq by auto*
moreover have $\nu s > \mu s \implies \nu (X t) > \mu (X t)$
using *assms(2) Xhyp thyp Ghyp unfolding diff-invariant-eq by auto*
ultimately show $\nu (X t) = \mu (X t) \implies \text{False}$
by auto
qed

lemma *diff-invariant-neq-rule-converse*:

fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
assumes $Uhyp: \bigwedge s. s \in S \Longrightarrow \text{is-interval } (U\ s) \bigwedge s\ t. s \in S \Longrightarrow t \in U\ s \Longrightarrow t_0 \leq t$
and $conts: \bigwedge X. (D\ X = (\lambda\tau. f\ \tau\ (X\ \tau))\ \text{on } U(X\ t_0)) \Longrightarrow \text{continuous-on } (\mathcal{P}\ X\ (U\ (X\ t_0)))\ \nu$
 $\bigwedge X. (D\ X = (\lambda\tau. f\ \tau\ (X\ \tau))\ \text{on } U(X\ t_0)) \Longrightarrow \text{continuous-on } (\mathcal{P}\ X\ (U\ (X\ t_0)))\ \mu$
and $dI:\text{diff-invariant } (\lambda s. \nu\ s \neq \mu\ s)\ f\ U\ S\ t_0\ G$
shows $\text{diff-invariant } (\lambda s. \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
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 $xivp: D\ X = (\lambda x. f\ x\ (X\ x))\ \text{on } U\ (X\ t_0)\ \nu\ (X\ t_0) < \mu\ (X\ t_0)\ X \in U\ (X\ t_0) \rightarrow S$
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 $dI\ tHyp\ xivp(2)\ Ghyp\ \text{ivp-solsI}[of\ X\ f\ U\ X\ t_0, OF\ xivp(1) - xivp(3)\ t0Hyp]$
unfolding *diff-invariant-eq by force*
moreover
{assume $\text{ineq2}:\nu\ (X\ t) > \mu\ (X\ t)$
note $\text{continuous-on-compose}[OF\ \text{vderiv-on-continuous-on}[OF\ xivp(1)]]$
hence $\text{continuous-on } (U\ (X\ t_0))\ (\nu \circ X)$ **and** $\text{continuous-on } (U\ (X\ t_0))\ (\mu \circ X)$
using $xivp(1)\ conts$ **by** *blast+*
also have $\{t_0--t\} \subseteq U\ (X\ t_0)$
using $\text{closed-segment-subset-interval}[OF\ Uhyp(1)\ t0Hyp\ tHyp]\ xivp(3)\ t0Hyp$
by *auto*
ultimately have $\text{continuous-on } \{t_0--t\}\ (\lambda\tau. \nu\ (X\ \tau))$
and $\text{continuous-on } \{t_0--t\}\ (\lambda\tau. \mu\ (X\ \tau))$
using $\text{continuous-on-subset}$ **by** *auto*
then obtain τ **where** $\tau \in \{t_0--t\}\ \mu\ (X\ \tau) = \nu\ (X\ \tau)$
using $\text{IVT-two-functions-real-ivl}[OF\ -\ -\ xivp(2)\ \text{ineq2}]$ **by** *force*
hence $\forall r \in \text{down } (U\ (X\ t_0))\ \tau. G\ (X\ r)$ **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 $dI\ tHyp\ xivp(2)\ \text{ivp-solsI}[of\ X\ f\ U\ X\ t_0, OF\ xivp(1) - xivp(3)\ t0Hyp]$
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 $\text{diff-invariant } I_1\ f\ U\ S\ t_0\ G$
and $\text{diff-invariant } I_2\ f\ U\ S\ t_0\ G$
shows $\text{diff-invariant } (\lambda s. I_1\ s \wedge I_2\ s)\ f\ U\ S\ t_0\ G$

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*

3.3 Picard-Lindelof

A locale with the assumptions of Picard-Lindelof's theorem. It extends *ll-on-open-it* by providing an initial time $t_0 \in T$.

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

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

lemma *flow-funcset-ex-ivl*:
assumes $s \in S$
shows $\text{flow } t_0 s \in \text{ex-ivl } s \rightarrow S$
using *flow-usolves-ode*[*OF* *init-time* $\langle s \in S \rangle$]
unfolding *usolves-ode-from-def solves-ode-def* **by** *blast*

lemma *flow-in-ivp-sols-ex-ivl*:
assumes $s \in S$

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

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

lemma subset-ex-ivlI :

$Y_1 \in \text{Sols } f \ (\lambda s. T) \ S \ t_0 \ s \implies \{t_0--t\} \subseteq T \implies A \subseteq \{t_0--t\} \implies A \subseteq \text{ex-ivl}$
 s

apply($\text{clarsimp simp: existence-ivl-def}$)
apply($\text{subgoal-tac } t_0 \in T, \text{clarsimp simp: csols-eq}$)
apply($\text{rule-tac } x=Y_1 \ \text{in } \text{exI}, \text{rule-tac } x=t \ \text{in } \text{exI}, \text{safe}, \text{force}$)
by ($\text{rule in-ivp-sols-subset}[\text{where } T=\lambda s. T], \text{auto}$)

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 $\text{is-interval } U$ **and** $U \subseteq \text{ex-ivl } s$
and $\text{xivp}: D \ Y_1 = (\lambda t. f \ t \ (Y_1 \ t)) \ \text{on } U \ Y_1 \ t_0 = s \ Y_1 \in U \rightarrow S$
and $\text{yivp}: D \ Y_2 = (\lambda t. f \ t \ (Y_2 \ t)) \ \text{on } U \ Y_2 \ t_0 = s \ Y_2 \in U \rightarrow S$
shows $Y_1 \ t = Y_2 \ t$

proof—

have $t_0 \in T$
using $\text{assms existence-ivl-subset}$ **by** auto
have $\text{key}: (\text{flow } t_0 \ s \ \text{usolves-ode } f \ \text{from } t_0) \ (\text{ex-ivl } s) \ S$
using $\text{flow-usolves-ode}[OF \ \langle t_0 \in T \rangle \ \langle s \in S \rangle]$.
hence $\forall t \in U. Y_1 \ t = \text{flow } t_0 \ s \ t$
unfolding $\text{usolves-ode-from-def solves-ode-def}$ **apply** safe
by ($\text{erule-tac } x=Y_1 \ \text{in } \text{allE}, \text{erule-tac } x=U \ \text{in } \text{allE}, \text{auto simp: assms}$)
also have $\forall t \in U. Y_2 \ t = \text{flow } t_0 \ s \ t$
using $\text{key unfolding usolves-ode-from-def solves-ode-def}$ **apply** safe
by ($\text{erule-tac } x=Y_2 \ \text{in } \text{allE}, \text{erule-tac } x=U \ \text{in } \text{allE}, \text{auto simp: assms}$)
ultimately show $Y_1 \ t = Y_2 \ t$
using assms **by** auto

qed

Applications of lemma unique-solution :

lemma $\text{unique-solution-closed-ivl}$:

assumes $\text{xivp}: D \ X = (\lambda t. f \ t \ (X \ t)) \ \text{on } \{t_0--t\} \ X \ t_0 = s \ X \in \{t_0--t\} \rightarrow S$
and $t \in T$

and $\text{yivp}: D \ Y = (\lambda t. f \ t \ (Y \ t)) \ \text{on } \{t_0--t\} \ Y \ t_0 = s \ Y \in \{t_0--t\} \rightarrow S$ **and**
 $s \in S$

shows $X \ t = Y \ t$

apply($\text{rule } \text{unique-solution}[OF \ \langle s \in S \rangle, \text{of } \{t_0--t\}], \text{simp-all add: assms}$)

apply($\text{unfold existence-ivl-def csols-eq ivp-sols-def, clarsimp}$)

using $\text{xivp } \langle t \in T \rangle$ **by** blast

lemma *solution-eq-flow*:
assumes *xivp*: $D X = (\lambda t. f t (X t))$ on *ex-ivl s* $X t_0 = s$ $X \in \text{ex-ivl } s \rightarrow S$
and $t \in \text{ex-ivl } s$ **and** $s \in S$
shows $X t = \text{flow } t_0 s t$
apply(*rule unique-solution*[*OF* $\langle s \in S \rangle$ *init-time-ex-ivl* $\langle t \in \text{ex-ivl } s \rangle$])
using *flow-has-vderiv-on-ex-ivl flow-funcset-ex-ivl* $\langle s \in S \rangle$ **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*: $Y_1 \in \text{Sols } f U S t_0 s$ **and** *ivp2*: $Y_2 \in \text{Sols } f U S t_0 s$
shows $Y_1 t = Y_2 t$
proof(*rule unique-solution*[*OF* $\langle s \in S \rangle$, *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*] $\langle t \in U s \rangle$ **by** *blast*
moreover **have** *obs1*: $Y_1 \in \text{Sols } f (\lambda s. \{t_0--t\}) S t_0 s$
by (*rule in-ivp-sols-subset*[*OF* - *calculation*(1) *ivp1*], *simp*)
moreover **have** *obs2*: $Y_2 \in \text{Sols } f (\lambda s. \{t_0--t\}) S t_0 s$
by (*rule in-ivp-sols-subset*[*OF* - *calculation*(1) *ivp2*], *simp*)
ultimately **show** $\{t_0--t\} \subseteq \text{ex-ivl } s$
apply(*unfold existence-ivl-def csols-eq, clarsimp*)
apply(*rule-tac* $x=Y_1$ **in** *exI*, *rule-tac* $x=t$ **in** *exI*)
using $\langle t \in U s \rangle$ **and** $\langle U s \subseteq T \rangle$ **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 $Y_1 \in \{t_0--t\} \rightarrow S$ **and** $Y_2 \in \{t_0--t\} \rightarrow S$
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*: $Y \in \text{Sols } f U S t_0 s$
shows *g-orbital* $f G U S t_0 s = \text{g-orbit } Y G (U s)$
proof–
have *eq1*: $\forall Z \in \text{Sols } f U S t_0 s. \forall t \in U s. Z t = Y t$
by (*clarsimp, rule ivp-unique-solution*[*OF* *assms*(1,2,3) - - *ivp*], *auto*)
have *g-orbital* $f G U S t_0 s \subseteq \text{g-orbit } (\lambda t. Y t) G (U s)$
proof
fix x **assume** $x \in \text{g-orbital } f G U S t_0 s$
then **obtain** Z **and** t
where *z-def*: $x = Z t \wedge t \in U s \wedge (\forall \tau \in \text{down } (U s) t. G (Z \tau)) \wedge Z \in \text{Sols } f U S t_0 s$
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  $x \in g\text{-orbit } Y G (U s)$ 
    using z-def eq1 unfolding g-orbit-eq by simp metis
qed
moreover have  $g\text{-orbit } Y G (U s) \subseteq g\text{-orbital } f G U S t_0 s$ 
  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  $f1 f2 :: \text{real} \Rightarrow 'a :: \text{banach} \Rightarrow 'a$ 
  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  $\varepsilon_1 L1$  where  $\varepsilon_1 > 0$  and  $L1: \bigwedge \tau. \tau \in \text{cball } t \varepsilon_1 \cap T \implies L1\text{-lipschitz-on}$ 
    ( $\text{cball } s \varepsilon_1 \cap S$ ) ( $f1 \tau$ )
    using local-lipschitzE[OF assms(1)  $\langle t \in T \rangle \langle s \in S \rangle$ ] by blast
  obtain  $\varepsilon_2 L2$  where  $\varepsilon_2 > 0$  and  $L2: \bigwedge \tau. \tau \in \text{cball } t \varepsilon_2 \cap T \implies L2\text{-lipschitz-on}$ 
    ( $\text{cball } s \varepsilon_2 \cap S$ ) ( $f2 \tau$ )
    using local-lipschitzE[OF assms(2)  $\langle t \in T \rangle \langle s \in S \rangle$ ] by blast
  have ballH:  $\text{cball } s (\min \varepsilon_1 \varepsilon_2) \cap S \subseteq \text{cball } s \varepsilon_1 \cap S \cap \text{cball } s (\min \varepsilon_1 \varepsilon_2) \cap S \subseteq$ 
     $\text{cball } s \varepsilon_2 \cap S$ 
    by auto
  have obs1:  $\forall \tau \in \text{cball } t \varepsilon_1 \cap T. L1\text{-lipschitz-on } (\text{cball } s (\min \varepsilon_1 \varepsilon_2) \cap S) (f1 \tau)$ 
    using lipschitz-on-subset[OF L1 ballH(1)] by blast
  also have obs2:  $\forall \tau \in \text{cball } t \varepsilon_2 \cap T. L2\text{-lipschitz-on } (\text{cball } s (\min \varepsilon_1 \varepsilon_2) \cap S)$ 
    ( $f2 \tau$ )
    using lipschitz-on-subset[OF L2 ballH(2)] by blast
  ultimately have  $\forall \tau \in \text{cball } t (\min \varepsilon_1 \varepsilon_2) \cap T. (L1 + L2)\text{-lipschitz-on } (\text{cball } s (\min \varepsilon_1 \varepsilon_2) \cap S) (\lambda s. f1 \tau s + f2 \tau s)$ 
    using lipschitz-on-add by fastforce
  thus  $\exists u > 0. \exists L. \forall t \in \text{cball } t u \cap T. L\text{-lipschitz-on } (\text{cball } s u \cap S) (\lambda s. f1 t s +$ 
     $f2 t s)$ 
    apply (rule-tac  $x=\min \varepsilon_1 \varepsilon_2$  in exI)
    using  $\langle \varepsilon_1 > 0 \rangle \langle \varepsilon_2 > 0 \rangle$  by force
qed

lemma picard-lindeloeff-add:  $\text{picard-lindeloeff } f1 T S t_0 \implies \text{picard-lindeloeff } f2 T S$ 
 $t_0 \implies$ 

```

```

picard-lindeloeff ( $\lambda t s. f1 t s + f2 t s$ )  $T S t_0$ 
unfolding picard-lindeloeff-def apply(clarsimp, rule conjI)
using continuous-on-add apply fastforce
using local-lipschitz-add by blast

```

```

lemma picard-lindeloeff-constant: picard-lindeloeff ( $\lambda t s. c$ )  $UNIV UNIV t_0$ 
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)

```

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-lindeloeff ( $\lambda t. f$ )  $T S 0$ 
for  $f::'a::\{heine-borel,banach\} \Rightarrow 'a$  and  $T S L +$ 
fixes  $\varphi :: real \Rightarrow 'a \Rightarrow 'a$ 
assumes ivp:
 $\bigwedge t s. t \in T \implies s \in S \implies D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s))$  on  $\{0--t\}$ 
 $\bigwedge s. s \in S \implies \varphi 0 s = s$ 
 $\bigwedge t s. t \in T \implies s \in S \implies (\lambda t. \varphi t s) \in \{0--t\} \rightarrow S$ 
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  $\langle t \in T \rangle$ ])
using  $\langle s \in S \rangle$  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) $\langle t \in T \rangle$* **by** *blast*
moreover have $t + r/2 > 0$
using $\langle r > 0 \rangle \langle t > 0 \rangle$ **by** *auto*
moreover have $\{0--t\} \subseteq T$
using *subintervalI[OF init-time $\langle t \in T \rangle$]* .
ultimately have *subs: $\{0<--<t + r/2\} \subseteq T$*
unfolding *abs-le-eq abs-le-eq real-ivl-eqs[OF $\langle t > 0 \rangle$] real-ivl-eqs[OF $\langle t + r/2 > 0 \rangle$]*
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 - $\langle s \in S \rangle$]* **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 $\langle t + r/2 > 0 \rangle$]* **by** *auto*
have $t \in \{0<--<t + r/2\}$
unfolding *real-ivl-eqs[OF $\langle t + r/2 > 0 \rangle$]* **using** *rHyp $\langle t > 0 \rangle$* **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 $\langle t + r/2 > 0 \rangle$]* **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 $r::\text{real}$ **where** *rHyp: $r > 0$ ball $t r \subseteq T$*

using *open-contains-ball-eq open-domain(1) $\langle t \in T \rangle$* **by** *blast*

moreover have $t - r/2 < 0$

using $\langle r > 0 \rangle \langle t < 0 \rangle$ **by** *auto*

moreover have $\{0--t\} \subseteq T$

using *subintervalI[OF init-time $\langle t \in T \rangle$]* .

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 $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash (t - r/2)\})$
 using $\text{ivp}(1)[OF - \langle s \in S \rangle]$ by *auto*
 hence $\text{vderiv}: (D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash \langle t - r/2 \rangle\})$
 apply $(\text{rule has-vderiv-on-subset})$
 unfolding $\text{open-segment-eq-real-ivl closed-segment-eq-real-ivl}$ by *auto*
 have $t \in \{0 \dashv\vdash \langle t - r/2 \rangle\}$
 unfolding $\text{open-segment-eq-real-ivl}$ using $rHyp \langle t < 0 \rangle$ by *simp*
 moreover have $D (\lambda \tau. \varphi \tau s) \mapsto (\lambda \tau. \tau *_R f (\varphi t s))$ (at t within $\{0 \dashv\vdash \langle t - r/2 \rangle\}$)
 using $\text{vderiv calculation unfolding has-vderiv-on-def has-vector-derivative-def}$
 by *blast*
 moreover have $\text{open } \{0 \dashv\vdash \langle t - r/2 \rangle\}$
 unfolding $\text{open-segment-eq-real-ivl}$ by *simp*
 ultimately show *?thesis*
 using $\text{subs that by blast}$
 qed

lemma *has-derivative-on-open3*:

assumes $s \in S$
 obtains B where $0 \in B$ and $\text{open } B$ 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 $r::\text{real}$ where $rHyp: r > 0$ ball $0 r \subseteq T$
 using $\text{open-contains-ball-eq open-domain}(1)$ *init-time* by *blast*
 hence $r/2 \in T -r/2 \in T r/2 > 0$
 unfolding real-ivl-eqs by *auto*
 hence $\text{subs}: \{0 \dashv\vdash r/2\} \subseteq T \{0 \dashv\vdash (-r/2)\} \subseteq T$
 using $\text{subintervalI}[OF \text{init-time}]$ by *auto*
 hence $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash r/2\})$
 $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash (-r/2)\})$
 using $\text{ivp}(1)[OF - \langle s \in S \rangle]$ by *auto*
 also have $\{0 \dashv\vdash r/2\} = \{0 \dashv\vdash r/2\} \cup \text{closure } \{0 \dashv\vdash r/2\} \cap \text{closure } \{0 \dashv\vdash (-r/2)\}$
 $\{0 \dashv\vdash (-r/2)\} = \{0 \dashv\vdash (-r/2)\} \cup \text{closure } \{0 \dashv\vdash r/2\} \cap \text{closure } \{0 \dashv\vdash (-r/2)\}$
 unfolding $\text{closed-segment-eq-real-ivl } \langle r/2 > 0 \rangle$ by *auto*
 ultimately have vderivs :
 $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash r/2\} \cup \text{closure } \{0 \dashv\vdash r/2\} \cap \text{closure } \{0 \dashv\vdash (-r/2)\})$
 $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{0 \dashv\vdash (-r/2)\} \cup \text{closure } \{0 \dashv\vdash r/2\} \cap \text{closure } \{0 \dashv\vdash (-r/2)\})$
 unfolding $\text{closed-segment-eq-real-ivl } \langle r/2 > 0 \rangle$ by *auto*
 have $\text{obs}: 0 \in \{-r/2 \dashv\vdash \langle r/2 \rangle\}$
 unfolding $\text{open-segment-eq-real-ivl}$ using $\langle r/2 > 0 \rangle$ by *auto*
 have $\text{union}: \{-r/2 \dashv\vdash \langle r/2 \rangle\} = \{0 \dashv\vdash r/2\} \cup \{0 \dashv\vdash (-r/2)\}$
 unfolding $\text{closed-segment-eq-real-ivl}$ by *auto*
 hence $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{-r/2 \dashv\vdash \langle r/2 \rangle\})$
 using $\text{has-vderiv-on-union}[OF \text{vderivs}]$ by *simp*
 hence $(D (\lambda t. \varphi t s) = (\lambda t. f (\varphi t s)) \text{ on } \{-r/2 \dashv\vdash \langle r/2 \rangle\})$
 using $\text{has-vderiv-on-subset}[OF - \text{segment-open-subset-closed}[of $-r/2 r/2$]]$ by *auto*

hence $D (\lambda\tau. \varphi \tau s) \mapsto (\lambda\tau. \tau *_R f (\varphi 0 s))$ (at 0 within $\{-r/2 <-- <r/2\}$)
unfolding *has-vderiv-on-def has-vector-derivative-def* **using** *obs* **by** *blast*
moreover have $\text{open } \{-r/2 <-- <r/2\}$
unfolding *open-segment-eq-real-ivl* **by** *simp*
moreover have $\{-r/2 <-- <r/2\} \subseteq T$
using *subs union segment-open-subset-closed* **by** *blast*
ultimately show *?thesis*
using *obs that* **by** *blast*
qed

lemma *has-derivative-on-open*:

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

lemma *in-domain*:

assumes $s \in S$
shows $(\lambda t. \varphi t s) \in T \rightarrow S$
using *ivp(3)*[*OF - assms*] **by** *blast*

lemma *has-vderiv-on-domain*:

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

lemma *in-ivp-sols*:

assumes $s \in S$ **and** $0 \in U$ s **and** $U \subseteq T$
shows $(\lambda t. \varphi t s) \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s$
apply(*rule in-ivp-sols-subset*[*OF - - ivp-solsI, of - - λs. T*])
using *ivp(2)*[*OF <s ∈ S>*] *has-vderiv-on-domain*[*OF <s ∈ S>*]
in-domain[*OF <s ∈ S>*] *assms* **by** *auto*

lemma *eq-solution*:

assumes $s \in S$ **and** *is-interval* $(U \ s)$ **and** $U \subseteq T$ **and** $t \in U$ s
and *xivp*: $X \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s$
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 $g\text{-orbital} (\lambda t. f) G U S 0 s = \{\varphi t s \mid t. t \in U s \wedge (\forall \tau \in \text{down} (U s) t. G (\varphi \tau s))\}$

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 (γ^φ)

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 $\gamma^\varphi s = \{\varphi t s \mid t. t \in T\}$

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 $g\text{-orbit} (\lambda t. \varphi t s) (\lambda s. True) T = \gamma^\varphi s$

```

unfolding g-orbit-eq orbit-eq[OF assms] by simp

end

lemma line-is-local-flow:
   $0 \in T \implies \text{is-interval } T \implies \text{open } T \implies \text{local-flow } (\lambda s. c) T \text{ UNIV } (\lambda t s. s + t *_{\mathbb{R}} c)$ 
  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

```

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 ( $\mu$ )
  and g-orbital ((1x' = - & - on - - @ -))

```

4.1 Verification of regular programs

Lemmas for verification condition generation

```

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:  $P \leq Q \implies |F| P \leq |F| Q$ 
  unfolding fbox-def by auto

```

```

lemma fbox-anti:  $\forall s. F_1 s \subseteq F_2 s \implies |F_2| P \leq |F_1| P$ 
  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*

— Skip

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

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

— Tests

definition *test* $:: 'a \text{ pred} \Rightarrow 'a \Rightarrow 'a \text{ set } ((1 \dot{!} - ?))$
where $\dot{!}P? = (\lambda s. \{x. x = s \wedge P x\})$

lemma *fbox-test[simp]*: $(\lambda s. (|\dot{!}P?| Q) s) = (\lambda s. P s \longrightarrow Q s)$
unfolding *fbox-def test-def* **by** *simp*

— Assignments

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

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

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

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*

— Nondeterministic assignments

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

lemma *fbox-nondet-assign[simp]*: $|x ::= ?| P = (\lambda s. \forall k. P (\chi j. \text{if } j = x \text{ then } k \text{ else } s\$j))$
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 - \\$ j) in alle, auto*)

— Nondeterministic choice

lemma *fbox-choice*: $|(\lambda s. F s \cup G s)| P = (\lambda s. (|F| P) s \wedge (|G| P) s)$
unfolding *fbox-def* **by** *auto*

lemma *le-fbox-choice-iff*: $P \leq |(\lambda s. F s \cup G s)| Q \longleftrightarrow P \leq |F| Q \wedge P \leq |G| Q$
unfolding *fbox-def* **by** *auto*

— Sequential composition

definition *kcomp* :: $('a \Rightarrow 'b \text{ set}) \Rightarrow ('b \Rightarrow 'c \text{ set}) \Rightarrow ('a \Rightarrow 'c \text{ set})$ (**infixl** ; 75)
where

$F ; G = \mu \circ \mathcal{P} G \circ F$

lemma *kcomp-eq*: $(f ; g) x = \bigcup \{g y \mid y. y \in f x\}$
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 $P \leq |G| R \ R \leq |F| Q$

shows $P \leq |G ; F| Q$

apply(*subst fbox-kcomp*)

by (*rule order.trans[OF assms(1)]*) (*rule fbox-iso[OF assms(2)]*)

— Conditional statement

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

lemma *fbox-if-then-else[simp]*:

$|IF T THEN X ELSE Y| Q = (\lambda s. (T s \longrightarrow (|X| Q) s) \wedge (\neg T s \longrightarrow (|Y| Q) s))$

unfolding *fbox-def ifthenelse-def* **by** *auto*

lemma *hoare-if-then-else*:

assumes $(\lambda s. P s \wedge T s) \leq |X| Q$

and $(\lambda s. P s \wedge \neg T s) \leq |Y| Q$

shows $P \leq |IF T THEN X ELSE Y| Q$

using *assms* **unfolding** *fbox-def ifthenelse-def* **by** *auto*

— Finite iteration

definition *kpower* :: $('a \Rightarrow 'a \text{ set}) \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow 'a \text{ set})$
where $kpower f n = (\lambda s. ((); f \overset{\sim}{\sim} n) \text{ skip } s)$

lemma *kpower-base*:

shows $kpower f 0 s = \{s\}$ **and** $kpower f (\text{Suc } 0) s = f s$

unfolding *kpower-def* **by**(*auto simp: kcomp-eq*)

lemma *kpower-simp*: $kpower f (\text{Suc } n) s = (f ; kpower f n) s$
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* = $\bigcup \{kpower\ f\ n\ s \mid n. n \in UNIV\}$

lemma *kpower-inv*:
fixes *F* :: 'a \Rightarrow 'a set
assumes $\forall s. I\ s \longrightarrow (\forall s'. s' \in F\ s \longrightarrow I\ s')$
shows $\forall s. I\ s \longrightarrow (\forall s'. s' \in (kpower\ F\ n\ s) \longrightarrow I\ s')$
apply(*clarsimp, induct n*)
unfolding *kpower-base kpower-simp*
apply(*simp-all add: kcomp-eq, clarsimp*)
apply(*subgoal-tac I y, simp*)
using *assms* **by** *blast*

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

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

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

lemma *change-loopI*: $LOOP\ X\ INV\ G = LOOP\ X\ INV\ I$
unfolding *loopi-def* **by** *simp*

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

lemma *wp-loopI-break*:
 $P \leq |Y| I \Longrightarrow I \leq |X| I \Longrightarrow I \leq Q \Longrightarrow P \leq |Y ; (LOOP\ X\ INV\ I)| Q$

by (rule hoare-kcomp, force) (rule fbox-loopI, auto)

4.2 Verification of hybrid programs

Verification by providing evolution

definition $g\text{-evol} :: (('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \text{ pred} \Rightarrow ('b \Rightarrow 'a \text{ set}) \Rightarrow ('b \Rightarrow 'b \text{ set})$ (EVOL)

where $EVOL \varphi G U = (\lambda s. g\text{-orbit} (\lambda t. \varphi t s) G (U s))$

lemma $fbox\text{-}g\text{-evol}[simp]$:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$

shows $|EVOL \varphi G U| Q = (\lambda s. (\forall t \in U s. (\forall \tau \in \text{down} (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

unfolding $g\text{-evol-def} g\text{-orbit-eq} fbox\text{-def}$ by auto

Verification by providing solutions

lemma $fbox\text{-}g\text{-orbital}: |x' = f \ \& \ G \text{ on } U S @ t_0| Q =$

$(\lambda s. \forall X \in \text{Sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down} (U s) t. G (X \tau)) \longrightarrow Q (X t))$

unfolding $fbox\text{-def} g\text{-orbital-eq}$ by (auto simp: fun-eq-iff)

context $local\text{-flow}$

begin

lemma $fbox\text{-}g\text{-ode-subset}$:

assumes $\bigwedge s. s \in S \Longrightarrow 0 \in U s \wedge is\text{-interval} (U s) \wedge U s \subseteq T$

shows $|x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0| Q =$

$(\lambda s. s \in S \longrightarrow (\forall t \in (U s). (\forall \tau \in \text{down} (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

apply (unfold $fbox\text{-}g\text{-orbital} fun\text{-eq-iff}$)

apply (clarify, rule iffI; clarify)

apply (force simp: $in\text{-ivp-sols} assms$)

apply (frule $ivp\text{-solsD}(2)$, frule $ivp\text{-solsD}(3)$, frule $ivp\text{-solsD}(4)$)

apply (subgoal-tac $\forall \tau \in \text{down} (U x) t. X \tau = \varphi \tau x$)

apply (clarsimp, fastforce, rule ballI)

apply (rule $ivp\text{-unique-solution}[OF \text{---} in\text{-ivp-sols}]$)

using $assms$ by auto

lemma $fbox\text{-}g\text{-ode}: |x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) S @ 0| Q =$

$(\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down} T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

by (subst $fbox\text{-}g\text{-ode-subset}$, simp-all add: $init\text{-time} interval\text{-time}$)

lemma $fbox\text{-}g\text{-ode-ivl}: t \geq 0 \Longrightarrow t \in T \Longrightarrow |x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{0..t\}) S @ 0| Q =$

$(\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

apply (subst $fbox\text{-}g\text{-ode-subset}$, simp-all add: $subintervalI init\text{-time} real\text{-Icc-closed-segment}$)

by (auto simp: $closed\text{-segment-eq-real-ivl}$)

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

unfolding $orbit\text{-def} fbox\text{-}g\text{-ode}$ 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'* = - & - on - - @ - *DINV* -))
where (*x' = f* & *G on U S @ t₀* *DINV I*) = (*x' = f* & *G on U S @ t₀*)

lemma *fbox-g-orbital-guard*:

assumes *H* = ($\lambda s. G s \wedge Q s$)

shows [*x' = f* & *G on U S @ t₀*] *Q* = [*x' = f* & *G on U S @ t₀*] *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 @ t₀*] *I* **and** *I* \leq *Q*

shows *P* \leq [*x' = f* & *G on U S @ t₀*] *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 @ t₀*] *I*) = *diff-invariant I f U S t₀ 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 @ t₀*] *I*

shows *I* \leq [*x' = f* & *G on U S @ t₀*] *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 $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

shows *diff-invariant I* ($\lambda t. f$) *U S 0* ($\lambda s. True$) =

(($\lambda s. s \in S \implies I s$) = [*x' =* ($\lambda t. f$) & ($\lambda s. True$) on *U S @ 0*] ($\lambda s. s \in S \implies I s$))

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* ($\lambda s. True$) = ($\forall s. I s \implies \gamma^\varphi s \subseteq \{s. I s\}$)

unfolding *diff-inv-eq-inv-set orbit-def* **by** *simp*

end

lemma *fbox-g-odei*: $P \leq I \implies I \leq |x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0] \ I \implies (\lambda s. I \ s \ \wedge \ G \ s) \leq Q \implies$

$P \leq |x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0 \ \text{DINV } I] \ Q$

unfolding *g-ode-inv-def*

apply(*rule-tac* $b = |x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0] \ I$ **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*)

4.3 Derivation of the rules of dL

We derive rules of differential dynamic logic (dL). This allows the components to reason in the style of that logic.

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

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

lemma *diff-solve-axiom1*:

assumes *local-flow f UNIV UNIV φ*

shows $|x' = f \ \& \ G] \ Q =$

$(\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$

by (*subst local-flow.fbox-g-ode-subset[OF assms], auto*)

lemma *diff-solve-axiom2*:

fixes $c::'a::\{\text{heine-borel}, \text{banach}\}$

shows $|x' = (\lambda s. c) \ \& \ G] \ Q =$

$(\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G \ (s + \tau *_R c)) \longrightarrow Q \ (s + t *_R c))$

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 $P \leq |x' = f \ \& \ G] \ Q$

using *assms* **by** (*subst local-flow.fbox-g-ode-subset[OF assms(1)] auto*)

lemma *diff-weak-axiom1*: $(|x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0] \ G) \ s$

unfolding *fbox-def g-orbital-eq* **by** *auto*

lemma *diff-weak-axiom2*: $|x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0] \ Q = |x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0] \ (\lambda s. G \ s \longrightarrow Q \ s)$

unfolding *fbx-g-orbital image-def* **by** *force*

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

lemma *fbx-g-orbital-eq-univD*:

assumes $|x' = f \ \& \ G \text{ on } U S @ t_0] C = (\lambda s. \text{True})$
and $\forall \tau \in (\text{down } (U s) t). x \ \tau \in (x' = f \ \& \ G \text{ on } U S @ t_0) s$
shows $\forall \tau \in (\text{down } (U s) t). C (x \ \tau)$

proof

fix τ **assume** $\tau \in (\text{down } (U s) t)$
hence $x \ \tau \in (x' = f \ \& \ G \text{ on } U S @ t_0) s$
using *assms(2)* **by** *blast*
also have $\forall s'. s' \in (x' = f \ \& \ G \text{ on } U S @ t_0) s \longrightarrow C s'$
using *assms(1)* **unfolding** *fbx-def* **by** *meson*
ultimately show $C (x \ \tau)$
by *blast*

qed

lemma *diff-cut-axiom*:

assumes $|x' = f \ \& \ G \text{ on } U S @ t_0] C = (\lambda s. \text{True})$
shows $|x' = f \ \& \ G \text{ on } U S @ t_0] Q = |x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } U S @ t_0] Q$
proof(*rule-tac f= $\lambda x. |x|$ Q in HOL.arg-cong, rule ext, rule subset-antisym*)

fix s

{fix s' **assume** $s' \in (x' = f \ \& \ G \text{ on } U S @ t_0) s$
then obtain $\tau :: \text{real}$ **and** X **where** $x\text{-ivp}: X \in \text{Sols } f \ U S \ t_0 \ s$
and $X \ \tau = s'$ **and** $\tau \in U s$ **and** $\text{guard-}x:\mathcal{P} \ X \ (\text{down } (U s) \ \tau) \subseteq \{s. G s\}$
using *g-orbitalD[of s' f G U S t_0 s]* **by** *blast*
have $\forall t \in (\text{down } (U s) \ \tau). \mathcal{P} \ X \ (\text{down } (U s) \ t) \subseteq \{s. G s\}$
using *guard-x* **by** (*force simp: image-def*)
also have $\forall t \in (\text{down } (U s) \ \tau). t \in U s$
using $\langle \tau \in U s \rangle$ *closed-segment-subset-interval* **by** *auto*
ultimately have $\forall t \in (\text{down } (U s) \ \tau). X \ t \in (x' = f \ \& \ G \text{ on } U S @ t_0) s$
using *g-orbitalI[OF x-ivp]* **by** (*metis (mono-tags, lifting)*)
hence $\forall t \in (\text{down } (U s) \ \tau). C (X \ t)$
using *assms unfolding fbx-def* **by** *meson*
hence $s' \in (x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } U S @ t_0) s$
using *g-orbitalI[OF x-ivp $\tau \in U s$] guard-x $X \ \tau = s'$* **by** *fastforce*
thus $(x' = f \ \& \ G \text{ on } U S @ t_0) s \subseteq (x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } U S @ t_0) s$
by *blast*

next show $\bigwedge s. (x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } U S @ t_0) s \subseteq (x' = f \ \& \ G \text{ on } U S @ t_0) s$

by (*auto simp: g-orbital-eq*)

qed

lemma *diff-cut-rule*:

assumes *fbx-C*: $P \leq |x' = f \ \& \ G \text{ on } U S @ t_0] C$
and *fbx-Q*: $P \leq |x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } U S @ t_0] Q$
shows $P \leq |x' = f \ \& \ G \text{ on } U S @ t_0] Q$

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

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

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

```

    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  $P \leq [x' = f \ \&\ G\ \text{on}\ U\ S\ @\ t_0]\ Q$ 
  apply (rule fbox-g-orbital-inv[OF assms(1) - assms(3)])
  unfolding fbox-diff-inv using assms(2) .

```

end

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

4.4.1 Pendulum

The ODEs $x' = -y$ and $y' = x$ describe the circular motion of a mass attached to a string looked from above. We use $s1$ to represent the x-coordinate and $s2$ for the y-coordinate. We prove that this motion remains circular.

```

abbreviation fpend ::  $\text{real}^2 \Rightarrow \text{real}^2$  (f)
  where  $f\ s \equiv (\chi\ i.\ \text{if}\ i = 1\ \text{then}\ s2\ \text{else}\ -s1)$ 

```

```

abbreviation pend-flow ::  $\text{real} \Rightarrow \text{real}^2 \Rightarrow \text{real}^2$  ( $\varphi$ )
  where  $\varphi\ t\ s \equiv (\chi\ i.\ \text{if}\ i = 1\ \text{then}\ s1 * \cos\ t + s2 * \sin\ t\ \text{else}\ -s1 * \sin\ t + s2 * \cos\ t)$ 

```

— Verified with annotated dynamics.

```

lemma pendulum-dyn:  $(\lambda s.\ r^2 = (s1)^2 + (s2)^2) \leq [EVOL\ \varphi\ G\ T]\ (\lambda s.\ r^2 = (s1)^2 + (s2)^2)$ 
  by force

```

— Verified with differential invariants.

```

lemma pendulum-inv:  $(\lambda s.\ r^2 = (s1)^2 + (s2)^2) \leq [x' = f \ \&\ G]\ (\lambda s.\ r^2 = (s1)^2 + (s2)^2)$ 
  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)

4.4.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

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

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

— 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** $\langle 0 > g \rangle$ **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 *diff-invariant* ($\lambda s. 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0$) ($\lambda t. f g$)
($\lambda s. UNIV$) $S t_0 G$
by (*auto intro!*: *poly-derivatives diff-invariant-rules*)

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies$
($\lambda s. s\$1 = h \wedge s\$2 = 0$) \leq
|*LOOP* (
 $(x' = (f g) \ \& \ (\lambda s. s\$1 \geq 0)) \ \text{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*) *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*) *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** $\dots = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (*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*: $g < 0 \implies h \geq 0 \implies$
 $(\lambda s. s\$1 = h \wedge s\$2 = 0) \leq$
 $|LOOP ($
 $(EVOL (\varphi g) (\lambda s. s\$1 \geq 0) T) ;$
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$
by (*rule fbox-loopI (auto simp: bb-real-arith)*)

— Verified with the flow.

lemma *local-flow-ball*: *local-flow (f g) UNIV UNIV (φg)*
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*: $g < 0 \implies h \geq 0 \implies$
 $(\lambda s. s\$1 = h \wedge s\$2 = 0) \leq$
 $|LOOP ($
 $(x'=(\lambda t. f g) \ \& \ (\lambda s. s\$1 \geq 0) \text{ on } (\lambda s. UNIV) UNIV @ 0) ;$
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$
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)*

4.4.3 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 θ , 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, $\mathcal{3}$ is the temperature detected by the thermometer, and $\mathcal{4}$ states whether the heater is on ($s\mathcal{4} = 1$) or off ($s\mathcal{4} = 0$). We prove that the thermostat keeps the room's temperature between $Tmin$ and $Tmax$.

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

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

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\mathcal{1} - s_2\mathcal{1}|$

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 $|a * (s_1\mathcal{1} + -\ L) + -\ (a * (s_2\mathcal{1} + -\ L))| = a * |s_1\mathcal{1} + -\ s_2\mathcal{1}|$

using $a1$ **by** (*simp add: abs-mult*)

thus $|a * (s_2\mathcal{1} - L) - a * (s_1\mathcal{1} - L)| = a * |s_1\mathcal{1} - s_2\mathcal{1}|$

using $f2$ *minus-real-def* **by** *presburger*

qed

lemma *local-lipschitz-temp-dyn*:

assumes $0 < (a::real)$

shows *local-lipschitz UNIV UNIV* ($\lambda t::real.\ f\ a\ L$)

apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)

apply(*clarsimp, rule-tac x=1 in exI, clarsimp, rule-tac x=a in exI*)

using *assms*

apply(*simp add: norm-diff-temp-dyn*)

apply(*simp add: norm-vec-def L2-set-def, unfold UNIV-4, clarsimp*)

unfolding *real-sqrt-abs[symmetric]* **by** (*rule real-le-lsqrt*) *auto*

lemma *local-flow-temp*: $a > 0 \implies \text{local-flow } (f\ a\ L)\ UNIV\ UNIV\ (\varphi\ a\ L)$

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 ≤ exp (-a * t) exp (-a * t) ≤ 1*
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less, simp*)
thus $Tmin ≤ exp (-a * t) * T$
using *Thyps* **by** (*simp add: pos-divide-le-eq*)
show $exp (-a * t) * T ≤ Tmax$
using *Thyps mult-left-le-one-le[OF - exp-ge-zero obs(2), of T]*
less-eq-real-def order-trans-rules(23) **by** *blast*
qed

lemma *temp-dyn-up-real-arith:*

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

lemmas *fbox-temp-dyn = local-flow.fbox-g-ode-subset[OF local-flow-temp]*

lemma *thermostat:*

assumes $a > 0$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $(λs. Tmin ≤ s\$1 ∧ s\$1 ≤ Tmax ∧ s\$4 = 0) ≤$
 $|LOOP$
— control
 $((2 ::= (λs. 0));(3 ::= (λs. s\$1)));$
 $(IF (λs. s\$4 = 0 ∧ s\$3 ≤ Tmin + 1) THEN (4 ::= (λs.1)) ELSE$

```

(IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$ ) THEN ( $4 ::= (\lambda s. 0)$ ) ELSE skip));
— dynamics
(IF ( $\lambda s. s\$4 = 0$ ) THEN ( $x' = f a 0 \ \& \ (\lambda s. s\$2 \leq -(\ln(Tmin/s\$3))/a)$ )
ELSE ( $x' = f a L \ \& \ (\lambda s. s\$2 \leq -(\ln((L-Tmax)/(L-s\$3)))/a)$ ))
INV ( $\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)$ )
( $\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax$ )
apply(rule fbox-loopI, simp-all add: fbox-temp-dyn[OF assms(1)] le-fun-def)
using temp-dyn-up-real-arith[OF assms(1) - - assms(3), of Tmin]
and temp-dyn-down-real-arith[OF assms(1,2), of - Tmax] by auto

```

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

4.4.4 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $hmin \leq h \leq hmax$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

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

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

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

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

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

lemma local-flow-tank: local-flow ($f k$) UNIV UNIV (φk)
apply (unfold-locales, unfold local-lipschitz-def lipschitz-on-def, simp-all, clar-simp)
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 $0 \leq (\tau :: \text{real})$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau \in \{0.. \tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0.. \tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \implies hmin \leq (c_i - c_o) * \tau + y$
and $y \leq hmax \implies y - c_o * \tau \leq hmax$
apply(*simp-all add: field-simps le-divide-eq assms*)
using *assms apply (meson add-mono less-eq-real-def mult-left-mono)*
using *assms by (meson add-increasing2 less-eq-real-def mult-nonneg-nonneg)*

lemma *tank-flow*:

assumes $0 < c_o$ **and** $c_o < c_i$
shows $I \ hmin \ hmax \leq$
|*LOOP*
— control
 $((\mathcal{Q} ::= (\lambda s. 0)); (\mathcal{P} ::= (\lambda s. s\$1)));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) THEN (\mathcal{A} ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) THEN (\mathcal{A} ::= (\lambda s. 0)) ELSE skip));$
— dynamics
 $(IF (\lambda s. s\$4 = 0) THEN (x' = f (c_i - c_o) \ \& \ (G \ hmax \ (c_i - c_o)))$
 $ELSE (x' = f (-c_o) \ \& \ (G \ hmin \ (-c_o)))) \) \ INV \ I \ hmin \ hmax$
 $I \ hmin \ hmax$
apply(*rule fbox-loopI, simp-all add: le-fun-def*)
apply(*clarsimp simp: le-fun-def local-flow.fbox-g-ode-subset[OF local-flow-tank]*)
using *assms tank-arith[OF - assms] by auto*

no-notation *tank-vec-field* (*f*)

and *tank-flow* (φ)
and *tank-loop-inv* (*I*)
and *tank-diff-inv* (*dI*)
and *tank-guard* (*G*)

end

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
Transformer-Semantics.Kleisli-Quantaloid
../HS-ODEs

begin

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* ($((1x' = - \& - \text{on} - - @ -))$)

5.1 Verification of regular programs

Properties of the forward box operator.

lemma *fb_F F S* = $(\bigcap \circ \mathcal{P} (- \text{op}_K F)) (- S)$
unfolding *ffb-def map-dual-def dual-set-def klift-def* **by** *simp*

lemma *fb_F F S* = $\{s. F s \subseteq S\}$
by (*auto simp: ffb-def kop-def klift-def map-dual-def dual-set-def f2r-def r2f-def*)

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

lemma *ffb-iso: P \leq Q \Longrightarrow fb_F F P \leq fb_F F Q*
unfolding *ffb-eq* **by** *auto*

lemma *ffb-invariants:*
assumes $\{s. I s\} \leq \text{fb}_F F \{s. I s\}$ **and** $\{s. J s\} \leq \text{fb}_F F \{s. J s\}$
shows $\{s. I s \wedge J s\} \leq \text{fb}_F F \{s. I s \wedge J s\}$
and $\{s. I s \vee J s\} \leq \text{fb}_F F \{s. I s \vee J s\}$
using *assms unfolding ffb-eq* **by** *auto*

— Skip

lemma *ffb-skip[simp]: fb_F skip S* = *S*
unfolding *ffb-def* **by** (*simp add: kop-def klift-def map-dual-def*)

— Tests

definition *test* :: *'a pred \Rightarrow 'a \Rightarrow 'a set* ($((1 \dot{?})$)
where $\dot{?}P = (\lambda s. \{x. x = s \wedge P x\})$

lemma *ffb-test[simp]: fb_F $\dot{?}P Q$* = $\{s. P s \longrightarrow s \in Q\}$
unfolding *ffb-eq test-def* **by** *simp*

— Assignments

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

where $(x ::= e) = (\lambda s. \{vec\text{-upd } s \ x \ (e \ s)\})$

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

— Nondeterministic assignments

definition $nondet\text{-assign} :: 'n \Rightarrow 'a \hat{\ } n \Rightarrow ('a \hat{\ } n) \text{ set } ((\lambda s. (x ::= ?)) [70] 61)$

where $(x ::= ?) = (\lambda s. \{(vec\text{-upd } s \ x \ k) \mid k. \ True\})$

lemma $fbx\text{-nondet}\text{-assign}[simp]: fb_{\mathcal{F}} (x ::= ?) \ P = \{s. \forall k. (\chi \ j. \text{if } j = x \text{ then } k \text{ else } s\$j) \in P\}$

unfolding $ffb\text{-eq}$ $nondet\text{-assign}\text{-def}$ $vec\text{-upd}\text{-eq}$ **apply** $(simp \ add: \ fun\text{-eq}\text{-iff}, \ safe)$
by $(erule\text{-tac } x=(\chi \ j. \text{if } j = x \text{ then } k \text{ else } - \$ j) \text{ in } allE, \ auto)$

— Nondeterministic choice

lemma $ffb\text{-choice}: fb_{\mathcal{F}} (\lambda s. \ F \ s \cup \ G \ s) \ P = fb_{\mathcal{F}} \ F \ P \cap \ fb_{\mathcal{F}} \ G \ P$

unfolding $ffb\text{-eq}$ **by** $auto$

lemma $le\text{-ffb}\text{-choice}\text{-iff}: P \subseteq fb_{\mathcal{F}} (\lambda s. \ F \ s \cup \ G \ s) \ Q \iff P \subseteq fb_{\mathcal{F}} \ F \ Q \wedge P \subseteq fb_{\mathcal{F}} \ G \ Q$

unfolding $ffb\text{-eq}$ **by** $auto$

— Sequential composition

lemma $ffb\text{-kcomp}[simp]: fb_{\mathcal{F}} (G ; F) \ P = fb_{\mathcal{F}} \ G \ (fb_{\mathcal{F}} \ F \ P)$

unfolding $ffb\text{-eq}$ **by** $(auto \ simp: \ kcomp\text{-def})$

lemma $hoare\text{-kcomp}$:

assumes $P \leq fb_{\mathcal{F}} \ F \ R \ R \leq fb_{\mathcal{F}} \ G \ Q$

shows $P \leq fb_{\mathcal{F}} (F ; G) \ Q$

apply $(subst \ ffb\text{-kcomp})$

by $(rule \ order.trans[OF \ assms(1)]) \ (rule \ ffb\text{-iso}[OF \ assms(2)])$

— Conditional statement

definition $ifthenelse :: 'a \ pred \Rightarrow ('a \Rightarrow 'b \ \text{set}) \Rightarrow ('a \Rightarrow 'b \ \text{set}) \Rightarrow ('a \Rightarrow 'b \ \text{set})$

$(IF \ - \ THEN \ - \ ELSE \ - [64, 64, 64] 63)$ **where**

$IF \ P \ THEN \ X \ ELSE \ Y = (\lambda \ x. \ \text{if } P \ x \ \text{then } X \ x \ \text{else } Y \ x)$

lemma $ffb\text{-if}\text{-then}\text{-else}[simp]$:

$fb_{\mathcal{F}} (IF \ T \ THEN \ X \ ELSE \ Y) \ Q = \{s. \ T \ s \longrightarrow s \in fb_{\mathcal{F}} \ X \ Q\} \cap \{s. \ \neg \ T \ s \longrightarrow s \in fb_{\mathcal{F}} \ Y \ Q\}$

unfolding $ffb\text{-eq}$ $ifthenelse\text{-def}$ **by** $auto$

lemma $hoare\text{-if}\text{-then}\text{-else}$:

assumes $P \cap \{s. T s\} \leq fb_{\mathcal{F}} X Q$
and $P \cap \{s. \neg T s\} \leq fb_{\mathcal{F}} Y Q$
shows $P \leq fb_{\mathcal{F}} (IF T THEN X ELSE Y) Q$
using *assms*
apply(*subst ffb-eq*)
apply(*subst (asm) ffb-eq*)
unfolding *ifthenelse-def* **by** *auto*

— 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 *change-loopI*: $LOOP X INV G = LOOP X INV I$
unfolding *loopi-def* **by** *simp*

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 *ffb-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*)

5.2 Verification of hybrid programs

Verification by providing evolution

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

lemma $fbx\text{-}g\text{-evol}[simp]$:

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

Verification by providing solutions

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

context $local\text{-flow}$

begin

lemma $ffb\text{-}g\text{-ode-subset}$:

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

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

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

lemma $ffb\text{-orbit}$: $fb_{\mathcal{F}} \gamma^\varphi Q = \{s. s \in S \longrightarrow (\forall t \in T. \varphi t s \in Q)\}$
unfolding $orbit\text{-def } ffb\text{-}g\text{-ode}$ **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'=- & - on - - @ - DINV -))
where (x' = f & G on U S @ t0 DINV I) = (x' = f & G on U S @ t0)

lemma *ffb-g-orbital-guard*:
assumes H = (λ s. G s \wedge Q s)
shows fb_F (x' = f & G on U S @ t0) {s. Q s} = fb_F (x' = f & G on U S @ t0) {s. H s}
unfolding *ffb-g-orbital* **using** *assms* **by** *auto*

lemma *ffb-g-orbital-inv*:
assumes P \leq I **and** I \leq fb_F (x' = f & G on U S @ t0) I **and** I \leq Q
shows P \leq fb_F (x' = f & G on U S @ t0) Q
using *assms(1)*
apply(*rule order.trans*)
using *assms(2)*
apply(*rule order.trans*)
by (*rule ffb-iso[OF assms(3)]*)

lemma *ffb-diff-inv[simp]*:
 ({s. I s} \leq fb_F (x' = f & G on U S @ t0) {s. I s}) = *diff-invariant I f U S t0 G*
by (*auto simp: diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *bd-diff-inv*:
diff-invariant I f U S t0 G = (bd_F (x' = f & G on U S @ t0) {s. I s} \leq {s. I s})
unfolding *ffb-fbd-galois-var* **by** (*auto simp: diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *diff-inv-guard-ignore*:
assumes {s. I s} \leq fb_F (x' = f & (λ s. True) on U S @ t0) {s. I s}
shows {s. I s} \leq fb_F (x' = f & G on U S @ t0) {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 \implies 0 \in U s \wedge *is-interval* (U s) \wedge U s \subseteq T
shows *diff-invariant I* (λ t. f) U S 0 (λ s. True) =
 ({s. s \in S \longrightarrow I s} = fb_F (x' = (λ t. f) & (λ s. True) 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*:

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 *ffb-g-odei*: $P \leq \{s. I s\} \Longrightarrow \{s. I s\} \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \{s. I s\} \Longrightarrow$

$\{s. I s \wedge G s\} \leq Q \Longrightarrow P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0 \ \text{DINV} \ I) \ Q$

unfolding *g-ode-inv-def*

apply(*rule-tac* $b = \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \{s. I s\}$ 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*)

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* :: $((a::\text{banach}) \Rightarrow a) \Rightarrow a \ \text{pred} \Rightarrow a \Rightarrow a \ \text{set} \ ((1x' = - \ \& \ -))$

where $(x' = f \ \& \ G) \equiv (x' = (\lambda t. f) \ \& \ G \ \text{on} \ (\lambda s. \{t. t \geq 0\}) \ \text{UNIV} \ @ \ 0)$

abbreviation *g-dl-ode-inv* :: $((a::\text{banach}) \Rightarrow a) \Rightarrow a \ \text{pred} \Rightarrow a \ \text{pred} \Rightarrow a \Rightarrow a \ \text{set} \ ((1x' = - \ \& \ - \ \text{DINV} \ -))$

where $(x' = f \ \& \ G \ \text{DINV} \ I) \equiv (x' = (\lambda t. f) \ \& \ G \ \text{on} \ (\lambda s. \{t. t \geq 0\}) \ \text{UNIV} \ @ \ 0 \ \text{DINV} \ I)$

lemma *diff-solve-axiom1*:

assumes *local-flow* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$

shows $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G) \ Q =$

$\{s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow (\varphi \ t \ s) \in Q\}$

by (*subst local-flow.ffib-g-ode-subset[OF assms], auto*)

lemma *diff-solve-axiom2*:

fixes $c::a::\{\text{heine-borel}, \text{banach}\}$

shows $\text{fb}_{\mathcal{F}} (x' = (\lambda s. c) \ \& \ G) \ Q =$

$\{s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (s + \tau *_R c)) \longrightarrow (s + t *_R c) \in Q\}$

apply(*subst local-flow.ffib-g-ode-subset*[**where** $\varphi = (\lambda t \ s. s + t *_R c)$ **and** $T = \text{UNIV}$])

by (*rule line-is-local-flow, auto*)

lemma *diff-solve-rule*:

assumes *local-flow* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$

and $\forall s. s \in P \longrightarrow (\forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow (\varphi \ t \ s) \in Q)$

shows $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G) \ Q$

using *assms* **by**(*subst local-flow.ffib-g-ode-subset*[*OF assms(1)*]) *auto*

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

lemma *diff-weak-axiom2*: $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0) \ Q = \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0) \ \{s. \ G \ s \longrightarrow s \in Q\}$
unfolding *ffb-g-orbital image-def* **by** *force*

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

lemma *ffb-g-orbital-eq-univD*:
assumes $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{s. \ C \ s\} = UNIV$
and $\forall \tau \in (\text{down} (U \ s) \ t). \ x \ \tau \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$
shows $\forall \tau \in (\text{down} (U \ s) \ t). \ C \ (x \ \tau)$

proof

fix τ **assume** $\tau \in (\text{down} (U \ s) \ t)$
hence $x \ \tau \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$
using *assms(2)* **by** *blast*
also have $\forall y. \ y \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s \longrightarrow C \ y$
using *assms(1)* **unfolding** *ffb-eq* **by** *fastforce*
ultimately show $C \ (x \ \tau)$ **by** *blast*

qed

lemma *diff-cut-axiom*:
assumes $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{s. \ C \ s\} = UNIV$
shows $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ Q = \text{fb}_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ Q$

proof(*rule-tac f= λ x. fb $_{\mathcal{F}}$ x Q in HOL.arg-cong, rule ext, rule subset-antisym*)

fix s

{fix s' **assume** $s' \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$
then obtain $\tau::\text{real}$ **and** X **where** $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$
and $X \ \tau = s'$ **and** $\tau \in (U \ s)$ **and** $\text{guard-}x:\mathcal{P} \ X \ (\text{down} (U \ s) \ \tau) \subseteq \{s. \ G \ s\}$
using *g-orbitalD[of s' f G - S t₀ s]* **by** *blast*
have $\forall t \in (\text{down} (U \ s) \ \tau). \ \mathcal{P} \ X \ (\text{down} (U \ s) \ t) \subseteq \{s. \ G \ s\}$
using *guard-x* **by** (*force simp: image-def*)
also have $\forall t \in (\text{down} (U \ s) \ \tau). \ t \in (U \ s)$
using $\langle \tau \in (U \ s) \rangle$ *closed-segment-subset-interval* **by** *auto*
ultimately have $\forall t \in (\text{down} (U \ s) \ \tau). \ X \ t \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$
using *g-orbitalI[OF x-ivp]* **by** (*metis (mono-tags, lifting)*)
hence $\forall t \in (\text{down} (U \ s) \ \tau). \ C \ (X \ t)$
using *assms* **unfolding** *ffb-eq* **by** *fastforce*
hence $s' \in (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ s$
using *g-orbitalI[OF x-ivp $\langle \tau \in (U \ s) \rangle$ guard-x $\langle X \ \tau = s' \rangle$]*
unfolding *image-le-pred* **by** *fastforce*}

thus $(x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ s$
by *blast*

next show $\bigwedge s. (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$

by (*auto simp: g-orbital-eq*)
qed

lemma *diff-cut-rule*:

assumes *ffb-C*: $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{s. \ C \ s\}$
and *ffb-Q*: $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ Q$
shows $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ Q$
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*: $X \in \text{Sols} \ f \ U \ S \ t_0 \ s$
and *guard-x*: $\forall \tau. \ \tau \in (U \ s) \ \wedge \ \tau \leq t \ \longrightarrow \ G \ (X \ \tau)$
have $\forall \tau \in (\text{down} \ (U \ s) \ t). \ X \ \tau \in (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ s$
using *g-orbitalI[OF x-ivp] guard-x unfolding image-le-pred by auto*
hence $\forall \tau \in (\text{down} \ (U \ s) \ t). \ C \ (X \ \tau)$
using *ffb-C* $\langle s \in P \rangle$ **by** (*subst (asm) ffb-eq, auto*)
hence $X \ t \in (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ s$
using *guard-x* $\langle t \in (U \ s) \rangle$ **by** (*auto intro!: g-orbitalI x-ivp*)
thus $(X \ t) \in Q$
using $\langle s \in P \rangle$ *ffb-Q* **by** (*subst (asm) ffb-eq, 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 $s \in (\text{fb}_{\mathcal{F}} (x' = f \ \& \ G) \ \{s. \ I \ s\})$
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-lindelof* $(\lambda t. \ f) \ UNIV \ UNIV \ 0$
and $\bigwedge s. \ \{t::real. \ t \geq 0\} \subseteq \text{picard-lindelof.ex-ivl} \ (\lambda t. \ f) \ UNIV \ UNIV \ 0 \ s$
and *diff-invariant I* $(\lambda t. \ f) \ (\lambda s. \ \{t::real. \ t \geq 0\}) \ UNIV \ 0 \ G$
shows $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G) \ \{s. \ I \ s\} = \text{fb}_{\mathcal{F}} (\lambda s. \ \{x. \ s = x \ \wedge \ G \ s\}) \ \{s. \ I \ s\}$
proof(*unfold ffb-g-orbital, subst ffb-eq, clarsimp simp: set-eq-iff*)
fix *s*
let *?ex-ivl s* = *picard-lindelof.ex-ivl* $(\lambda t. \ f) \ UNIV \ UNIV \ 0 \ s$
let *?lhs s* =
 $\forall X \in \text{Sols} \ (\lambda t. \ f) \ (\lambda s. \ \{t. \ t \geq 0\}) \ UNIV \ 0 \ s. \ \forall t \geq 0. \ (\forall \tau. \ 0 \leq \tau \ \wedge \ \tau \leq t \ \longrightarrow \ G$
 $(X \ \tau)) \ \longrightarrow \ I \ (X \ t)$
obtain *X* **where** *xivp1*: $X \in \text{Sols} \ (\lambda t. \ f) \ (\lambda s. \ ?ex-ivl \ s) \ UNIV \ 0 \ s$
using *picard-lindelof.flow-in-ivp-sols-ex-ivl[OF assms(1)] by auto*
have *xivp2*: $X \in \text{Sols} \ (\lambda t. \ f) \ (\lambda s. \ \text{Collect} \ ((\leq) \ 0)) \ UNIV \ 0 \ s$
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  $\leq$  {s. I s} and diff-invariant I f U S t0 G and {s. I s}  $\leq$  Q
shows P  $\leq$  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

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

5.4.1 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 :: real2  $\Rightarrow$  real2 (f)
  where f s  $\equiv$  ( $\chi$  i. if i = 1 then s$2 else -s$1)

```

```

abbreviation pend-flow :: real  $\Rightarrow$  real2  $\Rightarrow$  real2 ( $\varphi$ )
  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 the dynamics

```

lemma pendulum-dyn: {s. r2 = (s$1)2 + (s$2)2}  $\leq$  fbF (EVOL  $\varphi$  G T) {s. r2 = (s$1)2 + (s$2)2}
  by force

```

— Verified with differential invariants.

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

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

5.4.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

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

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

— 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** $\langle 0 > g \rangle$ **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*: $g < 0 \implies h \geq 0 \implies$
 $\{s. s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$
(LOOP (
 $(x' = (f g) \ \& \ (\lambda s. s\$1 \geq 0) \ \text{DINV} \ (\lambda s. 2 * g * s\$1 - 2 * g * h - s\$2 * s\2
 $= 0))$
 $);$
 $(IF \ (\lambda s. s\$1 = 0) \ \text{THEN} \ (2 ::= (\lambda s. - s\$2)) \ \text{ELSE} \ \text{skip}))$
 $\text{INV} \ (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0))$
 $\{s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
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 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 $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
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) 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) 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:  $g < 0 \implies h \geq 0 \implies$ 
  { $s. s\$1 = h \wedge s\$2 = 0$ }  $\leq$  fbF
  (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 ( $\varphi$   $g$ )
  apply(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff,
clarsimp)
  apply(rule-tac x=1/2 in exI, clarsimp, rule-tac x=1 in exI)
  apply(simp add: dist-norm norm-vec-def L2-set-def UNIV-2)
  by (auto simp: forall-2 intro!: poly-derivatives)

```

```

lemma bouncing-ball-flow:  $g < 0 \implies h \geq 0 \implies$ 
  { $s. s\$1 = h \wedge s\$2 = 0$ }  $\leq$  fbF
  (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.ffib-g-ode[OF local-flow-ball])

```

```

no-notation fball ( $f$ )
  and ball-flow ( $\varphi$ )

```

5.4.3 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 θ , 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* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (f)$
where $f a L s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } -a * (s\$1 - L) \text{ else } 0))$

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

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f a L s_1 - f a L s_2\| = |a| * |s_1\$1 - s_2\$1|$

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*: $a > 0 \implies \text{local-flow } (f a L) \text{ UNIV UNIV } (\varphi a L)$

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*: $0 < Tmin \text{ } Tmin \leq T \text{ } T \leq Tmax$

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 *thyyps* **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 *Thyyps* **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 *Thyyps* **by** (*simp add: pos-divide-le-eq*)
show $\exp(-a * t) * T \leq Tmax$
using *Thyyps 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** *Thyyps*: $Tmin \leq T \leq Tmax$ $Tmax < (L::real)$
and *thyyps*: $0 \leq t \forall \tau \in \{0..t\}. \tau \leq -(\ln((L - Tmax) / (L - T)) / a)$
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 *thyyps* **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 *Thyyps* **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 *Thyyps* **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 *Thyyps and obs* **by** *auto*

qed

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

lemma *thermostat*:

assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $\{s. Tmin \leq s \wedge s \leq Tmax \wedge s' = 0\} \leq fb_{\mathcal{F}}$

```

(LOOP
  — control
  ((2 ::= (λs. 0));(3 ::= (λs. s$1)));
  (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN (4 ::= (λs.1)) ELSE
  (IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax - 1) THEN (4 ::= (λs.0)) ELSE skip));
  — dynamics
  (IF (λs. s$4 = 0) THEN (x'=(λt. f a 0) & (λs. s$2 ≤ - (ln (Tmin/s$3))/a)
  on (λs. {0..t}) UNIV @ 0)
  ELSE (x'=(λt. f a L) & (λs. s$2 ≤ - (ln ((L-Tmax)/(L-s$3)))/a) on (λs.
  {0..t}) UNIV @ 0)) )
  INV (λs. Tmin ≤ s$1 ∧ s$1 ≤ Tmax ∧ (s$4 = 0 ∨ s$4 = 1))
  {s. Tmin ≤ s$1 ∧ s$1 ≤ 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)

5.4.4 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $hmin \leq h \leq hmax$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on (s4 = 1$) or off (s4 = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

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

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

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

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

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

lemma *local-flow-tank: local-flow (f k) UNIV UNIV (φ k)*
apply (*unfold-locales, unfold local-lipschitz-def lipschitz-on-def, simp-all, clar-simp*)
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 $\forall \tau \in \{0.. \tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0.. \tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \implies hmin \leq (c_i - c_o) * \tau + y$
and $y \leq hmax \implies y - c_o * \tau \leq hmax$
apply(*simp-all add: field-simps le-divide-eq assms*)
using *assms apply (meson add-mono less-eq-real-def mult-left-mono)*
using *assms by (meson add-increasing2 less-eq-real-def mult-nonneg-nonneg)*

lemma *tank-flow:*

assumes $0 < c_o$ **and** $c_o < c_i$
shows *Collect (I hmin hmax) \leq fb \mathcal{F}*
(LOOP
 — control
 (($2 ::= (\lambda s. 0)$);($3 ::= (\lambda s. s\$1)$));
 (IF ($\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$) THEN ($4 ::= (\lambda s. 1)$) ELSE
 (IF ($\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$) THEN ($4 ::= (\lambda s. 0)$) ELSE skip));
 — dynamics
 (IF ($\lambda s. s\$4 = 0$) THEN ($x' = f (c_i - c_o) \ \& \ (G \ hmax \ (c_i - c_o))$)
 ELSE ($x' = f (-c_o) \ \& \ (G \ hmin \ (-c_o))$))) INV I hmin hmax
(Collect (I hmin hmax))
apply(*rule ffb-loopI, simp-all add: le-fun-def*)
apply(*clarsimp simp: le-fun-def local-flow.ffb-g-ode-subset[OF local-flow-tank]*)
using *assms tank-arith[OF - assms] by auto*

no-notation *tank-vec-field (f)*

and *tank-flow (φ)*
and *tank-loop-inv (I)*
and *tank-diff-inv (dI)*
and *tank-guard (G)*

end

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

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*: $d p \leq |x + y| q \iff (d p \leq |x| q) \wedge (d p \leq |y| q)$
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 $\dots = d\ p \cdot |x| \ q$

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

$while\ t\ do\ x\ inv\ i = (d\ t \cdot x)^* \cdot ad\ t$

lemma *fbox-frame*: $d\ p \cdot x \leq x \cdot d\ p \implies d\ q \leq |x| \ r \implies d\ p \cdot d\ q \leq |x| \ (d\ p \cdot d\ r)$

using *dual.mult-isol-var fbox-add1 fbox-demodalisation3 fbox-simp* **by** *auto*

lemma *fbox-shunt*: $d\ p \cdot d\ q \leq |x| \ t \iff d\ p \leq ad\ q + |x| \ t$

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 $d\ t \cdot x \leq x \cdot d\ p \implies d\ t \cdot x \cdot ad\ q \leq x \cdot ad\ q \cdot d\ p \cdot ad\ q$

by (*metis (full-types) a-comm-var a-mult-idem ads-d-def am2 ds.ddual.mult-assoc phl-export2*)

hence $d\ t \leq |x| \ p \implies d\ t \leq |x \cdot ad\ q| \ (d\ p \cdot ad\ q)$

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*: $d p \cdot d t \leq |x| p \implies d p \leq |(d t \cdot x)^* \cdot ad t| (d p \cdot ad t)$

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 $d p \leq |(d t \cdot x)^*| p$
 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 $d i \leq |(d t \cdot x)^* \cdot ad t| (d i \cdot ad t)$
 using *fbox-while assms by blast*
 also have $\dots \leq |(d t \cdot x)^* \cdot ad t| q$
 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*: $p \leq |x| p' \implies p' \leq |y| q \implies p \leq |x \cdot y| q$

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*:

$d p \leq |y| i \implies d i \cdot ad t \leq d q \implies d i \cdot d t \leq |x| i \implies d p \leq |y \cdot (while t do x inv i)| q$

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**

lemma $d p \leq |x| p \implies d p \leq |x^*| p$

using *fbox-star-induct-var* .

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

lemma *fbox-loopi-break*:

$p \leq |y| d i \implies d i \leq |x| i \implies d i \leq d q \implies p \leq |y \cdot (loop\ x\ inv\ i)| q$
by (*rule fbox-seq-var, force*) (*rule fbox-loopi, auto*)

— Invariants

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

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

lemma $(i \leq |x| i) \vee (j \leq |x| j) \implies (i + j) \leq |x| (i + j)$

oops

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

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

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

end

end

6.2 Relational model

In this subsection, we follow Gomes and Struth [4] and show that relations form Kleene algebras.

theory *HS-VC-KA-rel*

imports *Kleene-Algebra.Kleene-Algebra*

begin

context *dioid-one-zero*

begin

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

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

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

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

lemma *rel-star-def*: $X \wedge^* = (\cup n. \text{rel-diod.power } X \ n)$
 by (*simp add: power-is-relpow rtrancl-is-UN-relpow*)

lemma *rel-star-contl*: $X \ O \ Y \wedge^* = (\cup n. X \ O \ \text{rel-diod.power } Y \ n)$
 by (*metis rel-star-def relcomp-UNION-distrib*)

lemma *rel-star-contr*: $X \wedge^* \ O \ Y = (\cup n. (\text{rel-diod.power } X \ n) \ O \ Y)$
 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-diod.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-contr, metis (lifting) SUP-le-iff rel-diod.power-inductr)
qed

end

```

6.3 Verification of hybrid programs

We show that relations form an antidomain Kleene algebra. This 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 hybrid systems in this setting.

theory *HS-VC-MKA-rel*

imports

../HS-ODEs

HS-VC-MKA

../HS-VC-KA-rel

begin

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) $\{ \}$ (\subseteq) (\subset) *rtrancl*

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

6.3.1 Regular programs

Lemmas for manipulation of predicates in the relational model

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

no-notation *Archimedean-Field.ceiling* ($\lceil \cdot \rceil$)

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[-])) **where**
 $[P] = \{(s,s) \mid s. P s\}$

lemma *p2r-simps*[*simp*]:

$$[P] \leq [Q] = (\forall s. P s \longrightarrow Q s)$$

$$([P] = [Q]) = (\forall s. P s = Q s)$$

$$([P] ; [Q]) = [\lambda s. P s \wedge Q s]$$

$$([P] \cup [Q]) = [\lambda s. P s \vee Q s]$$

$$\text{rel-ad } [P] = [\lambda s. \neg P s]$$

$$\text{rel-aka.ads-d } [P] = [P]$$

unfolding *p2r-def rel-ad-def rel-aka.ads-d-def* **by** *auto*

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

Lemmas for verification condition generation

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

— Tests

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

— Assignments

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

lemma *wp-assign* [*simp*]: $wp (x ::= e) [Q] = [\lambda s. Q (\chi j. (((\$) s)(x := (e \ s))) j)]$
unfolding *wp-rel vec-upd-def assign-def* **by** (*auto simp: fun-upd-def*)

— Nondeterministic assignments

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

lemma *wp-nondet-assign*[*simp*]: $wp (x ::= ?) [P] = [\lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j)]$

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*)

— Nondeterministic choice

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

— Conditional statement

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 [P] X Y

— Finite iteration

abbreviation *loopi-sugar* :: 'a rel \Rightarrow 'a pred \Rightarrow 'a rel (LOOP - INV - [64,64] 63)
where LOOP R INV I \equiv rel-aka.aka-loopi R [I]

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

lemma *wp-loopI*:
 $[P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow [I] \leq \text{wp } R [I] \Longrightarrow [P] \leq \text{wp } (\text{LOOP } R \text{ INV } I) [Q]$
using rel-aka.fbox-loopi[of [P]] **by** auto

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

6.3.2 Evolution commands

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 φ G U = {(s,s') | s s'. s' \in g-orbit ($\lambda t. \varphi t s$) G (U s)}

lemma *wp-g-dyn[simp]*:
fixes φ :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b
shows wp (EVOL φ G U) [Q] = $[\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)]$
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 @ t₀) = {(s,s') | s s'. s' \in g-orbital f G U S t₀ s}

lemma *wp-g-orbital*: wp (x' = f & G on U S @ t₀) [Q] = $[\lambda s. \forall X \in \text{Sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t)]$
unfolding g-orbital-eq wp-rel ivp-sols-def g-ode-def **by** auto

context *local-flow*
begin

lemma *wp-g-ode-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge is\text{-interval } (U s) \wedge U s \subseteq T$
shows $wp (x' = (\lambda t. f) \ \& \ G \text{ on } U S \ @ \ 0) [Q] =$
 $[\lambda s. s \in S \longrightarrow (\forall t \in (U s). (\forall \tau \in \text{down } (U s) \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))]$
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 $\forall \tau \in \text{down } (U s) \ t. X \ \tau = \varphi \ \tau \ s$*)
apply(*clarsimp, fastforce, rule ballI*)
apply(*rule ivp-unique-solution[OF - - - - in-ivp-sols]*)
using *assms by auto*

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

lemma *wp-g-ode-ivl*: $t \geq 0 \implies t \in T \implies wp (x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{0..t\})$
 $S \ @ \ 0) [Q] =$
 $[\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))]$
apply(*subst 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*: $wp (\{(s, s') \mid s \ s'. \ s' \in \gamma^\varphi \ s\}) [Q] = [\lambda s. s \in S \longrightarrow (\forall t \in T. Q$
 $(\varphi \ t \ s))]$
unfolding *orbit-def wp-g-ode g-ode-def[symmetric]* **by** *auto*

end

Verification with differential invariants

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

lemma *wp-g-orbital-guard*:

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

lemma *wp-g-orbital-inv*:

assumes $[P] \leq [I]$ **and** $[I] \leq wp (x' = f \ \& \ G \text{ on } U S \ @ \ t_0) [I]$ **and** $[I] \leq$
 $[Q]$
shows $[P] \leq wp (x' = f \ \& \ G \text{ on } U S \ @ \ t_0) [Q]$
using *assms(1)*
apply(*rule order.trans*)
using *assms(2)*

apply(*rule order.trans*)
apply(*rule rel-aka.fbox-iso*)
using *assms*(β) **by** *auto*

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

context *local-flow*
begin

lemma *wp-diff-inv-eq*:
assumes $\bigwedge s. s \in S \implies 0 \in U \ s \ \wedge \ \text{is-interval } (U \ s) \ \wedge \ U \ s \subseteq T$
shows *diff-invariant* $I \ (\lambda t. \ f) \ U \ S \ 0 \ (\lambda s. \ \text{True}) =$
 $(\lceil \lambda s. s \in S \longrightarrow I \ s \rceil = \text{wp } (x' = (\lambda t. \ f) \ \& \ (\lambda s. \ \text{True}) \ \text{on } U \ S \ @ \ 0) \ \lceil \lambda s. s \in S$
 $\longrightarrow I \ s \rceil)$
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-invariant $I \ (\lambda t. \ f) \ (\lambda s. \ T) \ S \ 0 \ (\lambda s. \ \text{True}) = (\forall s. I \ s \longrightarrow \gamma^\varphi \ s \subseteq \{s. I \ s\})$
unfolding *diff-inv-eq-inv-set orbit-def* **by** (*auto simp: p2r-def*)

end

lemma *wp-g-odei*: $\lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \ \lceil I \rceil \implies$
 $\lceil \lambda s. I \ s \ \wedge \ G \ s \rceil \leq \lceil Q \rceil \implies$
 $\lceil P \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0 \ \text{DINV } I) \ \lceil Q \rceil$
unfolding *g-ode-inv-def*
apply(*rule-tac b=wp (x' = f & G on U S @ t_0) [I] 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*)

6.3.3 Derivation of the rules of dL

We derive rules of differential dynamic logic (dL). This allows the components to reason in the style of that logic.

abbreviation *g-dl-ode* :: ($'a :: \text{banach}$) $\Rightarrow 'a \Rightarrow 'a \ \text{pred} \Rightarrow 'a \ \text{rel} \ ((1x' = - \ \& \ -))$

where $(x' = f \ \& \ G) \equiv (x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{t. t \geq 0\}) \text{ UNIV } @ \ 0)$

abbreviation $g\text{-dl-ode-inv} :: (('a::\text{banach}) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel}$
 $((\lambda x'. - \ \& \ - \ \text{DINV } -))$

where $(x' = f \ \& \ G \ \text{DINV } I) \equiv (x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{t. t \geq 0\}) \text{ UNIV } @ \ 0 \ \text{DINV } I)$

lemma *diff-solve-axiom1*:

assumes *local-flow* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$

shows $\text{wp } (x' = f \ \& \ G) \ [\![Q]\!] =$

$[\![\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)]\!]]$

by (*subst local-flow.wp-g-ode-subset[OF assms], auto*)

lemma *diff-solve-axiom2*:

fixes $c::'a::\{\text{heine-borel}, \text{banach}\}$

shows $\text{wp } (x' = (\lambda s. c) \ \& \ G) \ [\![Q]\!] =$

$[\![\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (s + \tau *_R c)) \longrightarrow Q (s + t *_R c)]\!]]$

apply(*subst local-flow.wp-g-ode-subset[where $\varphi = (\lambda t s. s + t *_R c)$ and $T = \text{UNIV}$]*)

by (*rule line-is-local-flow, auto*)

lemma *diff-solve-rule*:

assumes *local-flow* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$

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 $[\![P]\!] \leq \text{wp } (x' = f \ \& \ G) \ [\![Q]\!]]$

using *assms* **by** (*subst local-flow.wp-g-ode-subset[OF assms(1)], auto*)

lemma *diff-weak-axiom1*: $\text{Id} \subseteq \text{wp } (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [\![G]\!]]$

unfolding *wp-rel g-ode-def g-orbital-eq* **by** *auto*

lemma *diff-weak-axiom2*:

$\text{wp } (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [\![Q]\!] = \text{wp } (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [\![\lambda s. G \ s \longrightarrow Q \ s]\!]]$

unfolding *wp-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*:

assumes $[\![G]\!] \leq [\![Q]\!]]$

shows $[\![P]\!] \leq \text{wp } (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [\![Q]\!]]$

using *assms* **by** (*auto simp: wp-g-orbital*)

lemma *wp-g-evol-IdD*:

assumes $\text{wp } (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [\![C]\!] = \text{Id}$

and $\forall \tau \in (\text{down } (U \ s) \ t). (s, x \ \tau) \in (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0)$

shows $\forall \tau \in (\text{down } (U \ s) \ t). C (x \ \tau)$

proof

fix τ **assume** $\tau \in (\text{down } (U \ s) \ t)$

hence $x \ \tau \in \text{g-orbital } f \ G \ U \ S \ t_0 \ s$

using *assms(2)* **unfolding** *g-ode-def* **by** *blast*

also have $\forall y. y \in (\text{g-orbital } f \ G \ U \ S \ t_0 \ s) \longrightarrow C \ y$

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 $wp (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [C] = Id$
 shows $wp (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [Q] = wp (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \text{ on } U \ S \ @ \ t_0) \ [Q]$
 proof(*rule-tac* $f = \lambda x. \ wp \ x \ [Q]$ in *HOL.arg-cong*, *rule subset-antisym*)
 show $(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \subseteq (x' = f \ \& \ \lambda s. \ G \ s \ \wedge \ C \ s \text{ on } U \ S \ @ \ t_0)$
 proof(*clarsimp simp: g-ode-def*)
 fix s and s' assume $s' \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$
 then obtain $\tau :: \text{real}$ and X where $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$
 and $X \ \tau = s'$ and $\tau \in U \ s$ and $\text{guard-}x: (\mathcal{P} \ X \ (\text{down } (U \ s) \ \tau) \subseteq \{s. \ G \ s\})$
 using *g-orbitalD*[*of s' f G - S t_0 s*] by *blast*
 have $\forall t \in (\text{down } (U \ s) \ \tau). \ \mathcal{P} \ X \ (\text{down } (U \ s) \ t) \subseteq \{s. \ G \ s\}$
 using *guard-x* by (*force simp: image-def*)
 also have $\forall t \in (\text{down } (U \ s) \ \tau). \ t \in U \ s$
 using $\langle \tau \in U \ s \rangle$ by *auto*
 ultimately have $\forall t \in (\text{down } (U \ s) \ \tau). \ X \ t \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$
 using *g-orbitalI*[*OF x-ivp*] by (*metis (mono-tags, lifting)*)
 hence $\forall t \in (\text{down } (U \ s) \ \tau). \ C \ (X \ t)$
 using *wp-g-evol-IdD*[*OF assms(1)*] **unfolding** *g-ode-def* by *blast*
 thus $s' \in g\text{-orbital } f \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ U \ S \ t_0 \ s$
 using *g-orbitalI*[*OF x-ivp*] $\langle \tau \in U \ s \rangle$ *guard-x* $\langle X \ \tau = s' \rangle$ by *fastforce*
 qed
 next show $(x' = f \ \& \ \lambda s. \ G \ s \ \wedge \ C \ s \text{ on } U \ S \ @ \ t_0) \subseteq (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0)$
 by (*auto simp: g-orbital-eq g-ode-def*)
 qed

lemma *diff-cut-rule*:

assumes $wp\text{-}C: [P] \leq wp (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [C]$
 and $wp\text{-}Q: [P] \leq wp (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \text{ on } U \ S \ @ \ t_0) \ [Q]$
 shows $[P] \leq wp (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [Q]$
 proof(*subst wp-rel*, *simp add: g-orbital-eq p2r-def g-ode-def*, *clarsimp*)
 fix $t :: \text{real}$ and $X :: \text{real} \Rightarrow 'a$ and s assume $P \ s$ and $t \in U \ s$
 and $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$
 and $\text{guard-}x: \forall x. \ x \in U \ s \ \wedge \ x \leq t \longrightarrow G \ (X \ x)$
 have $\forall t \in (\text{down } (U \ s) \ t). \ X \ t \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$
 using *g-orbitalI*[*OF x-ivp*] *guard-x* by *auto*
 hence $\forall t \in (\text{down } (U \ s) \ t). \ C \ (X \ t)$
 using *wp-C* $\langle P \ s \rangle$ by (*subst (asm) wp-rel*, *auto simp: g-ode-def*)
 hence $X \ t \in g\text{-orbital } f \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ U \ S \ t_0 \ s$
 using *guard-x* $\langle t \in U \ s \rangle$ by (*auto intro!: g-orbitalI x-ivp*)
 thus $Q \ (X \ t)$
 using $\langle P \ s \rangle$ *wp-Q* by (*subst (asm) wp-rel*) (*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 $(s, s) \in wp (x' = f \ \& \ G) \lceil I \rceil$
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-lindelof* $(\lambda t. f)$ *UNIV UNIV 0*
and $\bigwedge s. \{t::real. t \geq 0\} \subseteq \text{picard-lindelof.ex-ivl} (\lambda t. f)$ *UNIV UNIV 0 s*
and *diff-invariant* $I (\lambda t. f) (\lambda s. \{t::real. t \geq 0\})$ *UNIV 0 G*
shows $wp (x' = f \ \& \ G) \lceil I \rceil = wp \lceil G \rceil \lceil I \rceil$
proof(*unfold wp-g-orbital, subst wp-rel, clarsimp simp: fun-eq-iff*)
fix s
let $?ex\text{-ivl } s = \text{picard-lindelof.ex-ivl} (\lambda t. f)$ *UNIV UNIV 0 s*
let $?lhs s =$
 $\forall X \in \text{Sols} (\lambda t. f) (\lambda s. \{t. t \geq 0\})$ *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** $xivp1: X \in \text{Sols} (\lambda t. f) (\lambda s. ?ex\text{-ivl } s)$ *UNIV 0 s*
using *picard-lindelof.flow-in-ivp-sols-ex-ivl[OF assms(1)] by auto*
have $xivp2: X \in \text{Sols} (\lambda t. f) (\lambda s. \text{Collect} ((\leq) 0))$ *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 $\lceil P \rceil \leq wp (x' = f \ \& \ G \text{ on } U S @ t_0) \lceil Q \rceil$
apply(*rule wp-g-orbital-inv[OF assms(1) - assms(3)]*)
unfolding *wp-diff-inv using assms(2)* .

end

6.4 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
```

6.4.1 Pendulum

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

```
abbreviation fpend :: real^2 ⇒ real^2 (f)
where f s ≡ (χ i. if i = 1 then s$2 else -s$1)
```

```
abbreviation pend-flow :: real ⇒ real^2 ⇒ real^2 (φ)
where φ t s ≡ (χ 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:
  [λs. r2 = (s$1)2 + (s$2)2] ≤ wp (EVOL φ G T) [λs. r2 = (s$1)2 + (s$2)2]
by simp
```

— Verified with differential invariants.

```
lemma pendulum-inv:
  [λs. r2 = (s$1)2 + (s$2)2] ≤ wp (x' = f & G) [λs. r2 = (s$1)2 + (s$2)2]
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:
  [λs. r2 = (s$1)2 + (s$2)2] ≤ wp (x' = f & G) [λs. r2 = (s$1)2 + (s$2)2]
by (simp add: local-flow.wp-g-ode-subset[OF local-flow-pend])
```

```
no-notation fpend (f)
```

and *pend-flow* (φ)

6.4.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

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

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

— 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** $\langle 0 > g \rangle$ **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 $g < 0 \implies h \geq 0 \implies \lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq$

wp

(*LOOP*

$((x' = f g \ \&\ (\lambda s. s\$1 \geq 0)) \text{ DINV } (\lambda s. 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0));$

$(\text{IF } (\lambda s. s\$1 = 0) \text{ THEN } (2 ::= (\lambda s. - s\$2)) \text{ ELSE skip}))$

$\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)$

) [$\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h$]
apply(*rule wp-loopI, simp-all, force simp: bb-real-arith*)
by (*rule wp-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 $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
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, opaque-lifting) 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, opaque-lifting) 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** $\dots = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** $\dots = ?middle$)
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** $\dots = ?middle$
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 $g < 0 \implies h \geq 0 \implies$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$

```

(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 ( $\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$ ))
  [ $\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h$ ]
  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 $g < 0 \implies h \geq 0 \implies$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$
(*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 ($\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\2))
 [$\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h$]
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)

6.4.3 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 θ , 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 θ 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* $:: real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$ (f)
where f a L $s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } -a * (s\$1 - L) \text{ else } 0))$

abbreviation *temp-flow* $:: real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$ (φ)

where $\varphi a L t s \equiv (\chi i. \text{if } i = 1 \text{ then } - \exp(-a * t) * (L - s\$1) + L \text{ else } (\text{if } i = 2 \text{ then } t + s\$2 \text{ else } s\$i))$

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f a L s_1 - f a L s_2\| = |a| * |s_1\$1 - s_2\$1|$

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-lsqr*) *auto*

lemma *local-flow-temp*: $a > 0 \implies \text{local-flow } (f a L) \text{ UNIV UNIV } (\varphi a L)$

by (*unfold-locale, 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*: $0 < T_{min} \ T_{min} \leq T \ T \leq T_{max}$

and *thyps*: $0 \leq (t::real) \ \forall \tau \in \{0..t\}. \tau \leq -(\ln(T_{min} / T) / a)$

shows $T_{min} \leq \exp(-a * t) * T$ **and** $\exp(-a * t) * T \leq T_{max}$

proof—

have $0 \leq t \wedge t \leq -(\ln(T_{min} / T) / a)$

using *thyps* **by** *auto*

hence $\ln(T_{min} / T) \leq -a * t \wedge -a * t \leq 0$

using *assms(1) divide-le-cancel* **by** *fastforce*

also have $T_{min} / T > 0$

using *Thyps* **by** *auto*

ultimately have *obs*: $T_{min} / T \leq \exp(-a * t) \ \exp(-a * t) \leq 1$

using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less, simp*)

thus $T_{min} \leq \exp(-a * t) * T$

using *Thyps* **by** (*simp add: pos-divide-le-eq*)

show $\exp(-a * t) * T \leq T_{max}$

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*: $Tmin \leq T \leq Tmax$ $Tmax < (L::real)$
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-subset*[*OF local-flow-temp*]

lemma *thermostat*:

assumes $a > 0$ and $0 < Tmin$ and $Tmax < L$

shows $\lceil \lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$

(*LOOP*

– control

$((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$

$(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1) THEN (4 ::= (\lambda s. 1)) ELSE$

$(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip));$

– dynamics

$(IF (\lambda s. s\$4 = 0) THEN (x' = f a 0 \ \& \ (\lambda s. s\$2 \leq -(\ln(Tmin/s\$3))/a))$

$ELSE (x' = f a L \ \& \ (\lambda s. s\$2 \leq -(\ln((L-Tmax)/(L-s\$3))/a)))$)

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)*])

using *temp-dyn-up-real-arith*[*OF assms(1) - - assms(3), of Tmin*]

and *temp-dyn-down-real-arith*[*OF assms*(1,2), *of - Tmax*] by *auto*

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

6.4.4 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $hmin \leq h \leq hmax$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

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

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

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

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

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

lemma *local-flow-tank*: *local-flow* (*f k*) *UNIV UNIV* ($\varphi\ k$)
apply (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clar-simp*)
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 $\forall \tau \in \{0.. \tau\}.\ \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0.. \tau\}.\ \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \implies hmin \leq (c_i - c_o) * \tau + y$
and $y \leq hmax \implies y - c_o * \tau \leq hmax$
apply(*simp-all add: field-simps le-divide-eq assms*)

using *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)
using *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

lemma *tank-flow*:

assumes $0 < c_o$ **and** $c_o < c_i$

shows $\lceil \lambda s. I \text{ hmin } \text{hmax } s \rceil \leq wp$

(*LOOP*

— control

$((\mathcal{Q} ::= (\lambda s. 0)); (\mathcal{P} ::= (\lambda s. s\$1)));$

(*IF* $(\lambda s. s\$4 = 0 \wedge s\$3 \leq \text{hmin} + 1)$ *THEN* $(4 ::= (\lambda s. 1))$ *ELSE*

(*IF* $(\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{hmax} - 1)$ *THEN* $(4 ::= (\lambda s. 0))$ *ELSE skip*));

— dynamics

(*IF* $(\lambda s. s\$4 = 0)$ *THEN* $(x' = f (c_i - c_o) \ \& \ (G \ \text{hmax} \ (c_i - c_o)))$

ELSE $(x' = f (-c_o) \ \& \ (G \ \text{hmin} \ (-c_o)))$) *INV* $I \ \text{hmin} \ \text{hmax}$)

$\lceil \lambda s. I \ \text{hmin} \ \text{hmax} \ s \rceil$

apply(*rule wp-loopI, simp-all add: local-flow.wp-g-ode-subset[OF local-flow-tank]*)

using *assms tank-arith[OF - assms]* **by** *auto*

no-notation *tank-vec-field* (*f*)

and *tank-flow* (φ)

and *tank-loop-inv* (*I*)

and *tank-diff-inv* (*dI*)

and *tank-guard* (*G*)

end

6.5 State transformer model

We show that Kleene algebras have a state transformer model. For this we use the type of non-deterministic functions of the `Transformer_Semantics.Kleisli_Quantale` theory. Below we prove some auxiliary lemmas for them and show this instantiation.

theory *HS-VC-KA-ndfun*

imports

Kleene-Algebra.Kleene-Algebra

Transformer-Semantics.Kleisli-Quantale

begin

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

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

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

lemma *nd-fun-ext*: $(\bigwedge x. (f\bullet) x = (g\bullet) x) \implies f = g$

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) = (∀ x. (f•) x = (g•) x)
  by (auto simp: nd-fun-ext)

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

definition 0 = ζ•

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

definition f + g = ((f•) ⊔ (g•))•

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

named-theorems nd-fun-ka kleene algebra properties for nondeterministic functions.

lemma nd-fun-plus-assoc[nd-fun-ka]: x + y + z = x + (y + z)
  and nd-fun-plus-comm[nd-fun-ka]: x + y = y + x
  and nd-fun-plus-idem[nd-fun-ka]: 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-ka]: (x + y) · z = x · z + y · z
  and nd-fun-distl[nd-fun-ka]: x · (y + z) = x · y + x · 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-zero1[nd-fun-ka]: 0 + x = x
  and nd-fun-mult-zero1[nd-fun-ka]: 0 · x = 0
  and nd-fun-mult-zero2[nd-fun-ka]: x · 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-ka]: (x ≤ y) = (x + y = y)
  and nd-fun-less[nd-fun-ka]: (x < y) = (x + y = y ∧ x ≠ y)
  and nd-fun-leq-add[nd-fun-ka]: z · x ≤ z · (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-ka]: 1 + x · x* ≤ x*
  and nd-star-unfoldl[nd-fun-ka]: z + x · y ≤ y ⇒ x* · z ≤ y
  and nd-star-unfoldr[nd-fun-ka]: z + y · x ≤ y ⇒ z · x* ≤ 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-ka by simp-all

```

end

end

6.6 Verification of hybrid programs

We show that non-deterministic functions or state transformers form an antidomain Kleene algebra. We use this algebra's forward box operator to derive rules for weakest liberal preconditions (wlps) of regular 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

../HS-VC-KA-ndfun

begin

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

begin

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

lemma *nd-fun-ad-zero[nd-fun-ka]*: $ad\ x \cdot x = 0$

and *nd-fun-ad[nd-fun-ka]*: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$

and *nd-fun-ad-one[nd-fun-ka]*: $ad\ (ad\ x) + ad\ x = 1$ **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-ka* **by** *simp-all*

end

6.6.1 Regular programs

Now that we know that non-deterministic 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.

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

notation *fbox* (*wp*)

no-notation *Archimedean-Field.ceiling* ($\lceil \cdot \rceil$)
and *Relation.relcomp* (**infixl** ; 75)
and *Range-Semiring.antirange-semiring-class.ars-r* (r)
and *antidomain-semiringl.ads-d* (d)

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

lemma $p2ndf\text{-simps}[simp]$:
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P s \longrightarrow Q s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P s = Q s)$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P s \wedge Q s \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P s \vee Q s \rceil$
 $ad \lceil P \rceil = \lceil \lambda s. \neg P s \rceil$
 $d \lceil P \rceil = \lceil P \rceil \lceil P \rceil \leq \eta^\bullet$
unfolding *less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def ads-d-def*
by (*auto simp: nd-fun-eq-iff kcomp-def le-fun-def antidomain-op-nd-fun-def*)

Lemmas for verification condition generation

lemma $wp\text{-nd-fun}$: $wp F \lceil P \rceil = \lceil \lambda s. \forall s'. s' \in ((F \bullet) s) \longrightarrow P s' \rceil$
apply (*simp add: fbox-def antidomain-op-nd-fun-def*)
by (*rule nd-fun-ext, auto simp: Rep-comp-hom kcomp-prop*)

— Skip

abbreviation $skip :: 'a \text{ nd-fun}$
where $skip \equiv 1$

— Tests

lemma $wp\text{-test}[simp]$: $wp \lceil P \rceil \lceil Q \rceil = \lceil \lambda s. P s \longrightarrow Q s \rceil$
by (*subst wp-nd-fun, simp*)

— Assignments

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

lemma $wp\text{-assign}[simp]$: $wp (x ::= e) \lceil Q \rceil = \lceil \lambda s. Q (\chi j. (((\$) s)(x := (e \ s))) j) \rceil$
unfolding $wp\text{-nd-fun nd-fun-eq-iff vec-upd-def assign-def}$ **by** *auto*

— Nondeterministic assignments

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

lemma $wp\text{-nondet-assign}[simp]$: $wp (x ::= ?) \lceil P \rceil = \lceil \lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j) \rceil$
unfolding $wp\text{-nd-fun nondet-assign-def vec-upd-eq}$ **apply** (*clarsimp, safe*)

by (*erule-tac* $x=(\chi j. \text{if } j = x \text{ then } k \text{ else } s \ \$ j)$ **in** *allE*, *auto*)

— Nondeterministic choice

lemma *le-wp-choice-iff*: $[P] \leq wp (X + Y) [Q] \longleftrightarrow [P] \leq wp X [Q] \wedge [P] \leq wp Y [Q]$
using *le-fbox-choice-iff*[of $[P]$] **by** *simp*

— Sequential composition

abbreviation *seq-comp* :: $'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun}$ (**infixl** ; 75)
where $f ; g \equiv f \cdot g$

— Conditional statement

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

— Finite iteration

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

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

lemma *wp-loopI*: $[P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow [I] \leq wp R [I] \Longrightarrow [P] \leq wp (LOOP R INV I) [Q]$
using *fbox-loopi*[of $[P]$] **by** *auto*

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

6.6.2 Evolution commands

Verification by providing evolution

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

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

Verification by providing solutions

definition $g\text{-ode} :: (\text{real} \Rightarrow ('a :: \text{banach}) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow ('a \Rightarrow \text{real set}) \Rightarrow 'a \text{ set} \Rightarrow$

$\text{real} \Rightarrow 'a \text{ nd-fun } ((1x' = - \& - \text{ on } - - @ -))$

where $(x' = f \& G \text{ on } U S @ t_0) \equiv (\lambda s. g\text{-orbital } f G U S t_0 s)^\bullet$

lemma $wp\text{-}g\text{-orbital}: wp (x' = f \& G \text{ on } U S @ t_0) [Q] =$

$[\lambda s. \forall X \in \text{Sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t)]$

unfolding $g\text{-orbital-eq}(1) wp\text{-nd-fun } g\text{-ode-def}$ **by** $(\text{auto simp: fun-eq-iff})$

context $local\text{-flow}$

begin

lemma $wp\text{-}g\text{-ode-subset}:$

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

shows $wp (x' = (\lambda t. f) \& G \text{ on } U S @ 0) [Q] =$

$[\lambda s. s \in S \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))]$

apply $(\text{unfold } wp\text{-}g\text{-orbital}, \text{clarsimp}, \text{rule iffI}; \text{clarify})$

apply $(\text{force simp: in-ivp-sols assms})$

apply $(\text{frule ivp-solsD}(2), \text{frule ivp-solsD}(3), \text{frule ivp-solsD}(4))$

apply $(\text{subgoal-tac } \forall \tau \in \text{down } (U s) t. X \tau = \varphi \tau s)$

apply $(\text{clarsimp}, \text{fastforce}, \text{rule ballI})$

apply $(\text{rule ivp-unique-solution}[OF \text{---} \text{---} \text{---} \text{---} \text{in-ivp-sols}])$

using assms **by** auto

lemma $wp\text{-}g\text{-ode}: wp (x' = (\lambda t. f) \& G \text{ on } (\lambda s. T) S @ 0) [Q] =$

$[\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))]$

by $(\text{subst } wp\text{-}g\text{-ode-subset}, \text{simp-all add: init-time interval-time})$

lemma $wp\text{-}g\text{-ode-ivl}: t \geq 0 \implies t \in T \implies wp (x' = (\lambda t. f) \& G \text{ on } (\lambda s. \{0..t\}) S @ 0) [Q] =$

$[\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))]$

apply $(\text{subst } wp\text{-}g\text{-ode-subset}, \text{simp-all add: subintervalI init-time real-Icc-closed-segment})$

by $(\text{auto simp: closed-segment-eq-real-ivl})$

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

unfolding $\text{orbit-def } wp\text{-}g\text{-ode } g\text{-ode-def}[\text{symmetric}]$ **by** auto

end

Verification with differential invariants

definition $g\text{-ode-inv} :: (\text{real} \Rightarrow ('a :: \text{banach}) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow ('a \Rightarrow \text{real set}) \Rightarrow 'a \text{ set} \Rightarrow$

$\text{real} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ nd-fun } ((1x' = - \& - \text{ on } - - @ - \text{ DINV } -))$

where $(x' = f \& G \text{ on } U S @ t_0 \text{ DINV } I) = (x' = f \& G \text{ on } U S @ t_0)$

lemma $wp\text{-}g\text{-orbital-guard}:$

assumes $H = (\lambda s. G s \wedge Q s)$

shows $wp (x' = f \& G \text{ on } U S @ t_0) [Q] = wp (x' = f \& G \text{ on } U S @ t_0) [H]$

unfolding *wp-g-orbital* **using** *assms* **by** *auto*

lemma *wp-g-orbital-inv*:

assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil I \rceil$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil$
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]*: $(\lceil I \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil I \rceil) = \text{diff-invariant } I \ f \ U \ S \ t_0 \ G$

unfolding *diff-invariant-eq wp-g-orbital* **by**(*auto simp: fun-eq-iff*)

lemma *diff-inv-guard-ignore*:

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

context *local-flow*

begin

lemma *wp-diff-inv-eq*:

assumes $\bigwedge s. s \in S \implies 0 \in U \ s \ \wedge \ \text{is-interval } (U \ s) \ \wedge \ U \ s \subseteq T$
shows $\text{diff-invariant } I \ (\lambda t. \ f) \ U \ S \ 0 \ (\lambda s. \ \text{True}) =$
 $(\lceil \lambda s. s \in S \implies I \ s \rceil = \text{wp } (x' = (\lambda t. \ f) \ \& \ (\lambda s. \ \text{True}) \ \text{on } U \ S \ @ \ 0) \lceil \lambda s. s \in S \implies I \ s \rceil)$
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*:

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

end

lemma *wp-g-odei*: $\lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil I \rceil \implies \lceil \lambda s. I \ s \ \wedge \ G \ s \rceil \leq \lceil Q \rceil \implies$

$\lceil P \rceil \leq \text{wp } (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0 \ \text{DINV } I) \lceil Q \rceil$

unfolding *g-ode-inv-def*

apply(*rule-tac b=wp (x' = f & G on U S @ t_0) [I]*) **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*)

6.6.3 Derivation of the rules of dL

We derive rules of differential dynamic logic (dL). This allows the components to reason in the style of that logic.

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

abbreviation *g-dl-ode-inv* :: ($'a::\text{banach}$) $\Rightarrow'a$) $\Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ nd-fun } ((1x'=- \& - \text{ DINV } -))$

where $(x'=f \& G \text{ DINV } I) \equiv (x'=(\lambda t. f) \& G \text{ on } (\lambda s. \{t. t \geq 0\}) \text{ UNIV } @ 0 \text{ DINV } I)$

lemma *diff-solve-axiom1*:

assumes *local-flow* $f \text{ UNIV UNIV } \varphi$
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 $c::'a::\{\text{heine-borel}, \text{banach}\}$
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** $T=\text{UNIV}$])
by (*rule line-is-local-flow*, *auto*)

lemma *diff-solve-rule*:

assumes *local-flow* $f \text{ UNIV UNIV } \varphi$
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*: $\eta^\bullet \leq wp (x'=f \& G \text{ on } U S @ t_0) \lceil G \rceil$

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*:

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

lemma *diff-weak-rule*:

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

lemma *wp-g-orbit-IdD*:

assumes $wp (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [C] = \eta^\bullet$
and $\forall \tau \in (\text{down} \ (U \ s) \ t). \ x \ \tau \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
shows $\forall \tau \in (\text{down} \ (U \ s) \ t). \ C \ (x \ \tau)$

proof

fix τ **assume** $\tau \in (\text{down} \ (U \ s) \ t)$
hence $x \ \tau \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
using *assms(2)* **by** *blast*
also have $\forall y. \ y \in (\text{g-orbital} \ f \ G \ U \ S \ t_0 \ s) \ \longrightarrow \ C \ y$
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 $wp (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [C] = \eta^\bullet$
shows $wp (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [Q] = wp (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ [Q]$
proof (*rule-tac f= $\lambda x. \ wp \ x \ [Q]$ in HOL.arg-cong, rule nd-fun-ext, rule subset-antisym*)
fix s **show** $((x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \bullet) \ s \subseteq ((x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \bullet) \ s$

proof (*clarsimp simp: g-ode-def*)

fix s' **assume** $s' \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
then obtain $\tau :: \text{real}$ **and** X **where** $x\text{-ivp}: X \in \text{ivp-sols} \ f \ U \ S \ t_0 \ s$
and $X \ \tau = s'$ **and** $\tau \in U \ s$ **and** $\text{guard-x}: (\mathcal{P} \ X \ (\text{down} \ (U \ s) \ \tau) \subseteq \{s. \ G \ s\})$
using *g-orbitalD[of s' f G - S t_0 s]* **by** *blast*
have $\forall t \in (\text{down} \ (U \ s) \ \tau). \ \mathcal{P} \ X \ (\text{down} \ (U \ s) \ t) \subseteq \{s. \ G \ s\}$
using *guard-x* **by** (*force simp: image-def*)
also have $\forall t \in (\text{down} \ (U \ s) \ \tau). \ t \in (U \ s)$
using $\langle \tau \in (U \ s) \rangle$ **by** *auto*
ultimately have $\forall t \in (\text{down} \ (U \ s) \ \tau). \ X \ t \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
using *g-orbitalI[OF x-ivp]* **by** (*metis (mono-tags, lifting)*)
hence $\forall t \in (\text{down} \ (U \ s) \ \tau). \ C \ (X \ t)$
using *wp-g-orbit-IdD[OF assms(1)]* **by** *blast*
thus $s' \in \text{g-orbital} \ f \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ U \ S \ t_0 \ s$
using *g-orbitalI[OF x-ivp $\tau \in (U \ s)$] guard-x $X \ \tau = s'$* **by** *fastforce*

qed

next

fix s **show** $((x' = f \ \& \ \lambda s. \ G \ s \ \wedge \ C \ s \ \text{on} \ U \ S \ @ \ t_0) \bullet) \ s \subseteq ((x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \bullet) \ s$
by (*auto simp: g-orbital-eq g-ode-def*)

qed

lemma *diff-cut-rule*:

assumes $wp\text{-}C: [P] \leq wp (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [C]$
and $wp\text{-}Q: [P] \leq wp (x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ [Q]$
shows $[P] \leq wp (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [Q]$
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 $x\text{-ivp}: X \in \text{ivp-sols } f \ U \ S \ t_0 \ s$
and $\text{guard-}x: \forall x. x \in U s \wedge x \leq t \longrightarrow G (X x)$
have $\forall t \in (\text{down } (U s) \ t). X \ t \in \text{g-orbital } f \ G \ U \ S \ t_0 \ s$
using $\text{g-orbitalI}[OF \ x\text{-ivp}] \ \text{guard-}x$ **by** auto
hence $\forall t \in (\text{down } (U s) \ t). C (X t)$
using $\text{wp-}C \ \langle P \ s \rangle$ **by** $(\text{subst } (asm) \ \text{wp-nd-fun}, \ \text{auto } \text{simp}: \text{g-ode-def})$
hence $X \ t \in \text{g-orbital } f \ (\lambda s. G \ s \wedge C \ s) \ U \ S \ t_0 \ s$
using $\text{guard-}x \ \langle t \in (U s) \rangle$ **by** $(\text{auto } \text{intro}!: \text{g-orbitalI } x\text{-ivp})$
thus $Q (X t)$
using $\langle P \ s \rangle \ \text{wp-}Q$ **by** $(\text{subst } (asm) \ \text{wp-nd-fun}) \ (\text{auto } \text{simp}: \text{g-ode-def})$
qed

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

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

6.7 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

6.7.1 Pendulum

The ODEs $x' = -y$ and $y' = x$ describe the circular motion of a mass attached to a string looked from above. We use $s\$1$ to represent the x-coordinate and $s\$2$ for the y-coordinate. We prove that this motion remains circular.

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

abbreviation *pend-flow* $:: real \Rightarrow real^2 \Rightarrow real^2\ (\varphi)$
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*:

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

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

6.7.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

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

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

— 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** $\langle 0 > g \rangle$ **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 $h::\text{real}$
shows $g < 0 \implies h \geq 0 \implies [\lambda s. s\$1 = h \wedge s\$2 = 0] \leq$
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 ($\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0$)
) [\lambda s. 0 \leq s\\$1 \wedge s\\$1 \leq h]
apply(*rule wp-loopI, simp-all, force simp: bb-real-arith*)
by (*rule wp-g-odei*) (*auto intro!: poly-derivatives diff-invariant-rules*)

— Verified with annotated dynamics.

lemma *inv-conserv-at-ground*[*bb-real-arith*]:

assumes *invar*: $2 * g * x = 2 * g * h + v * v$
and *pos*: $g * \tau^2 / 2 + v * \tau + (x::\text{real}) = 0$
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) 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) 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 $2 * g * (g * \tau^2 / 2 + v * \tau + (x::\text{real})) =$
 $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v))$ (**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** $\dots = g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$ (**is** $\dots = ?middle$)
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 $g < 0 \implies h \geq 0 \implies$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$
(LOOP
((EVOL (φ g) ($\lambda s. 0 \leq s\$1$) T);
(IF ($\lambda s. s\$1 = 0$) THEN ($2 ::= (\lambda s. - s\$2)$) ELSE skip))
*INV ($\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\2)*
 $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
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 $g < 0 \implies h \geq 0 \implies$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$
(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)*
 $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
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)

6.7.3 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 ($s4 = 1$) or off ($s4 = 0$). We prove that the thermostat keeps the room's temperature between $Tmin$ and $Tmax$.

abbreviation *temp-vec-field* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (f)$
where $f a L s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } -a * (s1 - L) \text{ else } 0))$

abbreviation *temp-flow* :: $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (\varphi)$
where $\varphi a L t s \equiv (\chi i. \text{if } i = 1 \text{ then } -\exp(-a * t) * (L - s1) + L \text{ else } (\text{if } i = 2 \text{ then } t + s2 \text{ else } s3))$

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f a L s_1 - f a L s_2\| = |a| * |s_1 - s_2|$

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 $|a * (s_1 + - L) + - (a * (s_2 + - L))| = a * |s_1 + - s_2|$

using $a1$ **by** (*simp add: abs-mult*)

thus $|a * (s_2 - L) - a * (s_1 - L)| = a * |s_1 - s_2|$

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-lsqr*) *auto*

lemma *local-flow-temp*: $a > 0 \implies \text{local-flow } (f a L) \text{ UNIV UNIV } (\varphi a L)$

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*: $0 < T_{min} \ T_{min} \leq T \ T \leq T_{max}$
and *thyps*: $0 \leq (t::real) \ \forall \tau \in \{0..t\}. \ \tau \leq -(\ln (T_{min} / T) / a)$
shows $T_{min} \leq \exp(-a * t) * T$ **and** $\exp(-a * t) * T \leq T_{max}$

proof –

have $0 \leq t \wedge t \leq -(\ln (T_{min} / T) / a)$
using *thyps* **by** *auto*
hence $\ln (T_{min} / T) \leq -a * t \wedge -a * t \leq 0$
using *assms(1) divide-le-cancel* **by** *fastforce*
also have $T_{min} / T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $T_{min} / T \leq \exp(-a * t) \ \exp(-a * t) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less, simp*)
thus $T_{min} \leq \exp(-a * t) * T$
using *Thyps* **by** (*simp add: pos-divide-le-eq*)
show $\exp(-a * t) * T \leq T_{max}$
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*: $T_{min} \leq T \ T \leq T_{max} \ T_{max} < (L::real)$
and *thyps*: $0 \leq t \ \forall \tau \in \{0..t\}. \ \tau \leq -(\ln ((L - T_{max}) / (L - T)) / a)$
shows $L - T_{max} \leq \exp(-(a * t)) * (L - T)$
and $L - \exp(-(a * t)) * (L - T) \leq T_{max}$
and $T_{min} \leq L - \exp(-(a * t)) * (L - T)$

proof –

have $0 \leq t \wedge t \leq -(\ln ((L - T_{max}) / (L - T)) / a)$
using *thyps* **by** *auto*
hence $\ln ((L - T_{max}) / (L - T)) \leq -a * t \wedge -a * t \leq 0$
using *assms(1) divide-le-cancel* **by** *fastforce*
also have $(L - T_{max}) / (L - T) > 0$
using *Thyps* **by** *auto*
ultimately have $(L - T_{max}) / (L - T) \leq \exp(-a * t) \wedge \exp(-a * t) \leq 1$
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*: $(L - T_{max}) \leq \exp(-a * t) * (L - T) \wedge \exp(-a * t) * (L - T) \leq (L - T)$
by (*simp add: pos-divide-le-eq*)
thus $(L - T_{max}) \leq \exp(-a * t) * (L - T)$
by *auto*
thus $L - \exp(-a * t) * (L - T) \leq T_{max}$
by *auto*
show $T_{min} \leq L - \exp(-a * t) * (L - T)$
using *Thyps and obs* **by** *auto*

qed

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

lemma *thermostat*:

assumes $a > 0$ **and** $0 < Tmin$ **and** $Tmax < L$

shows $\lceil \lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$

(*LOOP*

— control

$((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$

$(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1) THEN (4 ::= (\lambda s. 1)) ELSE$

$(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip));$

— dynamics

$(IF (\lambda s. s\$4 = 0) THEN (x' = f a 0 \ \& \ (\lambda s. s\$2 \leq - (\ln (Tmin/s\$3))/a))$

$ELSE (x' = f a L \ \& \ (\lambda s. s\$2 \leq - (\ln ((L-Tmax)/(L-s\$3))/a)))$)

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)])

using *temp-dyn-up-real-arith*[*OF assms*(1) - - *assms*(3), *of Tmin*]

and *temp-dyn-down-real-arith*[*OF assms*(1,2), *of - Tmax*] **by** *auto*

no-notation *temp-vec-field* (f)

and *temp-flow* (φ)

6.7.4 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $hmin \leq h \leq hmax$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

abbreviation *tank-vec-field* :: $real \Rightarrow real^4 \Rightarrow real^4$ (f)

where $f k s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } k \text{ else } 0))$

abbreviation *tank-flow* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$ (φ)

where $\varphi k \tau s \equiv (\chi i. \text{if } i = 1 \text{ then } k * \tau + s\$1 \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

abbreviation *tank-guard* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$ (G)

where $G Hm k s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$ (I)

where $I hmin hmax s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *tank-diff-inv* :: $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$ (dI)

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

lemma *local-flow-tank: local-flow (f k) UNIV UNIV (φ k)*

apply (*unfold-locales, unfold local-lipschitz-def lipschitz-on-def, simp-all, clar-simp*)

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 $\forall \tau \in \{0.. \tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$

and $\forall \tau \in \{0.. \tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$

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

and $y \leq hmax \implies y - c_o * \tau \leq hmax$

apply(*simp-all add: field-simps le-divide-eq assms*)

using *assms apply (meson add-mono less-eq-real-def mult-left-mono)*

using *assms by (meson add-increasing2 less-eq-real-def mult-nonneg-nonneg)*

lemma *tank-flow:*

assumes $0 < c_o$ **and** $c_o < c_i$

shows $\lceil \lambda s. I hmin hmax s \rceil \leq wp$

(*LOOP*

— *control*

$((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$

$(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) THEN (4 ::= (\lambda s. 1)) ELSE$

$(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip));$

— *dynamics*

$(IF (\lambda s. s\$4 = 0) THEN (x' = f (c_i - c_o) \& (G hmax (c_i - c_o)))$

$ELSE (x' = f (-c_o) \& (G hmin (-c_o))))) INV I hmin hmax$

$\lceil \lambda s. I hmin hmax s \rceil$

apply(*rule wp-loopI, simp-all add: local-flow.wp-g-ode-subset[OF local-flow-tank]*)

using *assms tank-arith[OF - assms] by auto*

no-notation *tank-vec-field (f)*

and *tank-flow (φ)*

and *tank-loop-inv (I)*

and *tank-diff-inv (dI)*

and *tank-guard (G)*

end

7 Verification components with KAT

We use Kleene algebras with tests to derive rules for verification condition generation and refinement laws.

theory *HS-VC-KAT*

imports *KAT-and-DRA.PHL-KAT*

begin

7.1 Hoare logic in KAT

Here we derive the rules of Hoare Logic.

notation t (**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* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ (H) **where**
 $H\ p\ x\ q \iff \text{tt}\ p \cdot x \leq x \cdot \text{tt}\ q$

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

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

lemma *H-cons*: $\text{tt}\ p \leq \text{tt}\ p' \implies \text{tt}\ q' \leq \text{tt}\ q \implies H\ p'\ x\ q' \implies H\ p\ x\ q$
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*: $H\ p\ x\ r \implies H\ r\ y\ q \implies H\ p\ (x \cdot y)\ q$
by (*simp add: Hoare-def phl-seq*)

— Nondeterministic choice

lemma *H-choice*: $H p x q \implies H p y q \implies H p (x + y) q$
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 = $(\text{tt } p \cdot x + n p \cdot y)$

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

lemma *H-cond-iff*: $H p (\text{if } r \text{ then } x \text{ else } y) q \longleftrightarrow H (\text{tt } p \cdot \text{tt } r) x q \wedge H (\text{tt } p \cdot n r) y q$

proof –

have $H p (\text{if } r \text{ then } x \text{ else } y) q \longleftrightarrow \text{tt } p \cdot (\text{tt } r \cdot x + n r \cdot y) \cdot n q = 0$
by (*simp add: H-var kat-cond-def*)

also have $\dots \longleftrightarrow \text{tt } p \cdot \text{tt } r \cdot x \cdot n q + \text{tt } p \cdot n r \cdot y \cdot n q = 0$
by (*simp add: distrib-left mult-assoc*)

also have $\dots \longleftrightarrow \text{tt } p \cdot \text{tt } r \cdot x \cdot n q = 0 \wedge \text{tt } p \cdot n r \cdot y \cdot n q = 0$
by (*metis add-0-left no-trivial-inverse*)

finally show *?thesis*

by (*metis H-var test-mult*)

qed

lemma *H-cond*: $H (\text{tt } p \cdot \text{tt } r) x q \implies H (\text{tt } p \cdot n r) y q \implies H p (\text{if } r \text{ then } x \text{ else } y) q$
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 = $(\text{tt } b \cdot x)^* \cdot n b$

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

lemma *H-while*: $H (\text{tt } p \cdot \text{tt } r) x p \implies H p (\text{while } r \text{ do } x) (\text{tt } p \cdot n r)$

proof –

assume *a1*: $H (\text{tt } p \cdot \text{tt } r) x p$

have $\text{tt } (\text{tt } p \cdot n r) = n r \cdot \text{tt } p \cdot n r$

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*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \cdot n \ r \leq \text{tt } q \implies H (\text{tt } i \cdot \text{tt } r) \ x \ i \implies H \ p \ (\text{while } r \ \text{inv } i \ \text{do } x) \ q$

by (*metis H-cons H-while test-mult kat-while-inv-def*)

— Finite iteration

lemma *H-star*: $H \ i \ x \ i \implies H \ i \ (x^*) \ i$

unfolding *Hoare-def* **using** *star-sim2* **by** *blast*

lemma *H-star-inv*:

assumes $\text{tt } p \leq \text{tt } i$ **and** $H \ i \ x \ i$ **and** $(\text{tt } i) \leq (\text{tt } q)$

shows $H \ p \ (x^*) \ q$

proof —

have $H \ i \ (x^*) \ i$

using *assms(2)* *H-star* **by** *blast*

hence $H \ p \ (x^*) \ i$

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 $\text{loop } x \ \text{inv } i = x^*$

lemma *H-loop*: $H \ p \ x \ p \implies H \ p \ (\text{loop } x \ \text{inv } i) \ p$

unfolding *kat-loop-inv-def* **by** (*rule H-star*)

lemma *H-loop-inv*: $\text{tt } p \leq \text{tt } i \implies H \ i \ x \ i \implies \text{tt } i \leq \text{tt } q \implies H \ p \ (\text{loop } x \ \text{inv } i) \ q$

unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

lemma *H-inv*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies H \ i \ x \ i \implies H \ p \ x \ q$

by (*rule-tac p'=i and q'=i in H-cons*)

lemma *H-inv-plus*: $\text{tt } i = i \implies \text{tt } j = j \implies H \ i \ x \ i \implies H \ j \ x \ j \implies H \ (i + j) \ x \ (i + j)$

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*: $\text{tt } i = i \implies \text{tt } j = j \implies H \ i \ x \ i \implies H \ j \ x \ j \implies H \ (i \cdot j) \ x \ (i \cdot j)$

unfolding *Hoare-def* **by** (*smt n-kat-2 n-mult-comm t-mult-closure mult-assoc*)

end

7.2 refinement KAT

Here we derive the laws of the refinement calculus.

```

class rkat = kat +
  fixes Ref :: 'a ⇒ 'a ⇒ 'a
  assumes spec-def:  $x \leq \text{Ref } p \ q \longleftrightarrow H \ p \ x \ q$ 

begin

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

lemma R2:  $H \ p \ x \ q \implies x \leq \text{Ref } p \ q$ 
  by (simp add: spec-def)

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

— Skip

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

— Abort

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

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

```

qed

lemma *R-abort*: $0 \leq \text{Ref } p \ q$
using *bot-least* **by** *force*

— Sequential composition

lemma *R-seq*: $(\text{Ref } p \ r) \cdot (\text{Ref } r \ q) \leq \text{Ref } p \ q$

proof —

have $H \ p \ (\text{Ref } p \ r) \ r$ **and** $H \ r \ (\text{Ref } r \ q) \ q$
by (*simp add: R1*)
hence $H \ p \ ((\text{Ref } p \ r) \cdot (\text{Ref } r \ q)) \ q$
by (*rule H-seq*)
thus *?thesis*
by (*rule R2*)

qed

— Nondeterministic choice

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

— Conditional statement

lemma *R-cond*: *if* v *then* $(\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q)$ *else* $(\text{Ref } (n \ v \cdot \text{tt } p) \ q) \leq \text{Ref } p \ q$

proof —

have $H \ (\text{tt } v \cdot \text{tt } p) \ (\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q) \ q$ **and** $H \ (n \ v \cdot \text{tt } p) \ (\text{Ref } (n \ v \cdot \text{tt } p) \ q) \ q$
by (*simp add: R1*)
hence $H \ p \ (\text{if } v \text{ then } (\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q) \text{ else } (\text{Ref } (n \ v \cdot \text{tt } p) \ q)) \ q$
by (*simp add: H-cond n-mult-comm*)
thus *?thesis*
by (*rule R2*)

qed

— While loop

lemma *R-while*: *while* q *do* $(\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p) \leq \text{Ref } p \ (\text{tt } p \cdot n \ q)$

proof —

have $H \ (\text{tt } p \cdot \text{tt } q) \ (\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p) \ p$
by (*simp-all add: R1*)
hence $H \ p \ (\text{while } q \text{ do } (\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p)) \ (\text{tt } p \cdot n \ q)$
by (*simp add: H-while*)
thus *?thesis*
by (*rule R2*)

qed

— Finite iteration

lemma *R-star*: $(\text{Ref } i \ i)^* \leq \text{Ref } i \ i$

proof –

have $H \ i \ (\text{Ref } i \ i) \ i$
 using *R1* **by** *blast*
 hence $H \ i \ ((\text{Ref } i \ i)^*) \ i$
 using *H-star* **by** *blast*
 thus $\text{Ref } i \ i^* \leq \text{Ref } i \ i$
 by (*rule R2*)

qed

lemma *R-loop*: $\text{loop } (\text{Ref } p \ p) \ \text{inv } i \leq \text{Ref } p \ p$

unfolding *kat-loop-inv-def* **by** (*rule R-star*)

— Invariants

lemma *R-inv*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies \text{Ref } i \ i \leq \text{Ref } p \ q$

using *R-cons* **by** *force*

end

end

7.3 Verification of hybrid programs

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-ODEs*
 HS-VC-KAT
 ../*HS-VC-KA-rel*

begin

interpretation *rel-tests*: *test-semiring* $(\cup) \ (O) \ Id \ \{\} \ (\subseteq) \ (\subset) \ \lambda x. \ Id \ \cap \ (- \ x)$

by (*standard, auto*)

interpretation *rel-kat*: *kat* $(\cup) \ (O) \ Id \ \{\} \ (\subseteq) \ (\subset) \ rtrancl \ \lambda x. \ Id \ \cap \ (- \ x)$

by (*unfold-locales*)

definition *rel-R* :: $'a \ rel \Rightarrow 'a \ rel \Rightarrow 'a \ rel$ **where**

$rel-R \ P \ Q = \bigcup \{X. \ rel-kat.Hoare \ P \ X \ Q\}$

interpretation *rel-rkat*: *rkat* $(\cup) \ (O) \ Id \ \{\} \ (\subseteq) \ (\subset) \ rtrancl \ (\lambda X. \ Id \ \cap \ - \ X) \ rel-R$

by (*standard, auto simp: rel-R-def rel-kat.Hoare-def*)

7.3.1 Regular programs

Lemmas for manipulation of predicates in the relational model

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

notation Id (*skip*)
and $empty$ (*abort*)
and $relcomp$ (**infixr** ; 75)

no-notation $Archimedean-Field.ceiling$ ($\lceil \cdot \rceil$)
and $Archimedean-Field.floor-ceiling-class.floor$ ($\lfloor \cdot \rfloor$)
and tau (τ)
and $n-op$ ($n - [90] 91$)

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

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

Lemmas for verification condition generation

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

— Hoare Triples

abbreviation $relHoare$ ($\{-\}-\{-\}$)
where $\{P\}X\{Q\} \equiv rel-kat.Hoare \lceil P \rceil X \lceil Q \rceil$

lemma $rel-kat-H: \{P\} X \{Q\} \longleftrightarrow (\forall s s'. P s \longrightarrow (s,s') \in X \longrightarrow Q s')$
by (*simp add: rel-kat.Hoare-def, auto simp add: p2r-def*)

— Skip

lemma $sH-skip[simp]: \{P\} skip \{Q\} \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
unfolding $rel-kat-H$ **by** *simp*

lemma $H-skip: \{P\} skip \{P\}$
by *simp*

— Tests

lemma *sH-test*[simp]: $\{P\} [R] \{Q\} = (\forall s. P\ s \longrightarrow R\ s \longrightarrow Q\ s)$
by (*subst rel-kat-H*, *simp add: p2r-def*)

— Abort

lemma *sH-abort*[simp]: $\{P\} \text{abort } \{Q\} \longleftrightarrow \text{True}$
unfolding *rel-kat-H* **by** *simp*

lemma *H-abort*: $\{P\} \text{abort } \{Q\}$
by *simp*

— Assignments

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

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

lemma *H-assign*: $P = (\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow \{P\} (x ::= e) \{Q\}$
by *simp*

— Nondeterministic assignments

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

lemma *sH-nondet-assign*[simp]:
 $\{P\} (x ::= ?) \{Q\} \longleftrightarrow (\forall s. P\ s \longrightarrow (\forall k. Q\ (\chi\ j. (((\$)\ s)(x := k))\ j)))$
unfolding *rel-kat-H* *vec-upd-def* *nondet-assign-def* **by** (*auto simp: fun-upd-def*)

lemma *H-nondet-assign*: $\{\lambda s. \forall k. P\ (\chi\ j. (((\$)\ s)(x := k))\ j)\} (x ::= ?) \{P\}$
by *simp*

— Sequential Composition

lemma *H-seq*: $\{P\} X \{R\} \Longrightarrow \{R\} Y \{Q\} \Longrightarrow \{P\} X; Y \{Q\}$
using *rel-kat.H-seq* .

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

lemma *H-assignl*:
assumes $\{K\} X \{Q\}$
and $\forall s. P\ s \longrightarrow K\ (\text{vec-lambda } (((\$)\ s)(x := e\ s)))$
shows $\{P\} (x ::= e); X \{Q\}$
apply(*rule H-seq*, *subst sH-assign*)
using *assms* **by** *auto*

— Nondeterministic Choice

lemma *sH-choice*: $\{P\} X \cup Y \{Q\} \longleftrightarrow (\{P\} X \{Q\} \wedge \{P\} Y \{Q\})$
unfolding *rel-kat-H* **by** *auto*

lemma *H-choice*: $\{P\} X \{Q\} \Longrightarrow \{P\} Y \{Q\} \Longrightarrow \{P\} X \cup Y \{Q\}$
using *rel-kat.H-choice* .

— Conditional Statement

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

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

lemma *H-cond*:
 $\{\lambda s. P s \wedge B s\} X \{Q\} \Longrightarrow \{\lambda s. P s \wedge \neg B s\} Y \{Q\} \Longrightarrow \{P\} (\text{IF } B \text{ THEN } X \text{ ELSE } Y) \{Q\}$
by *simp*

— While Loop

abbreviation *while-inv-sugar* :: $'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel}$
(*WHILE - INV - DO - [64,64,64] 63*)
where *WHILE B INV I DO X* $\equiv \text{rel-kat.kat-while-inv } [B] [I] X$

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

lemma $\{\lambda s. P s \wedge B s\} X \{\lambda s. P s\} \Longrightarrow \{P\} (\text{WHILE } B \text{ INV } I \text{ DO } X) \{\lambda s. P s \wedge \neg B s\}$
using *rel-kat.H-while[of [P] [B] X]*
unfolding *rel-kat.kat-while-inv-def* **by** *auto*

— Finite Iteration

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

lemma *H-loop*: $\{P\} X \{P\} \Longrightarrow \{P\} (\text{LOOP } X \text{ INV } I) \{P\}$
by (*auto intro: rel-kat.H-loop*)

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

7.3.2 Evolution commands

definition *g-evol* :: $(('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \text{ pred} \Rightarrow ('b \Rightarrow 'a \text{ set}) \Rightarrow 'b \text{ rel}$
(*EVOL*)

where *EVOL* $\varphi G U = \{(s,s') \mid s s'. s' \in g\text{-orbit } (\lambda t. \varphi t s) G (U s)\}$

lemma *sH-g-evol[simp]*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$

shows $\{P\} (EVOL \varphi G U) \{Q\} = (\forall s. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

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 $P = (\lambda s. (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

shows $\{P\} (EVOL \varphi G U) \{Q\}$

by (*simp add: assms*)

— Verification by providing solutions

definition *g-ode* :: $(\text{real} \Rightarrow ('a::\text{banach}) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow ('a \Rightarrow \text{real set}) \Rightarrow 'a$
 $\text{set} \Rightarrow \text{real} \Rightarrow$

$'a \text{ rel } ((\lambda x' = - \& - \text{ on } - - @ -))$

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

lemma *H-g-orbital*:

$P = (\lambda s. (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))) \implies$

$\{P\} (x' = f \& G \text{ on } U S @ t_0) \{Q\}$

unfolding *rel-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

lemma *sH-g-orbital*: $\{P\} (x' = f \& G \text{ on } U S @ t_0) \{Q\} =$

$(\forall s. P s \longrightarrow (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t)))$

unfolding *g-orbital-eq g-ode-def rel-kat-H* **by** *auto*

context *local-flow*

begin

lemma *sH-g-ode-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

shows $\{P\} (x' = (\lambda t. f) \& G \text{ on } U S @ 0) \{Q\} =$

$(\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

proof(*unfold sH-g-orbital, clarsimp, safe*)

fix $s t$
assume $hyp_s: s \in S \ P \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G (\varphi \ \tau \ s)$
and $main: \forall s. P \ s \longrightarrow (\forall X \in Sols (\lambda t. f) \ U \ S \ 0 \ s. \forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G (X \ \tau)) \longrightarrow Q (X \ t))$
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 $hyp_s: P \ s \ X \in Sols (\lambda t. f) \ U \ S \ 0 \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G (X \ \tau)$
and $main: \forall s \in S. P \ s \longrightarrow (\forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))$
hence $obs: s \in S$
using *ivp-sols-def[of $\lambda t. f$] init-time by auto*
hence $\forall \tau \in down (U \ s) \ t. X \ \tau = \varphi \ \tau \ s$
using *eq-solution hyps assms by blast*
thus $Q (X \ t)$
using *hyps main obs by auto*
qed

lemma *H-g-ode-subset*:

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

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

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

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

end

— Verification with differential invariants

definition *g-ode-inv* :: $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow ('a \Rightarrow real \ set) \Rightarrow$

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

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

lemma *H-g-ode-inv*: $\{I\} (x' = f \& G \text{ on } U S @ t_0) \{I\} \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil \lambda s. I s \wedge G s \rceil \leq \lceil Q \rceil \Longrightarrow \{P\} (x' = f \& G \text{ on } U S @ t_0 \text{ DINV } I) \{Q\}$
 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*)

7.4 Refinement Components

lemma *R-skip*: $(\forall s. P s \longrightarrow Q s) \Longrightarrow \text{skip} \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 by (*rule rel-rkat.R2, simp*)

— Abort

lemma *R-abort*: $\text{abort} \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 by (*rule rel-rkat.R2, simp*)

— Sequential Composition

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

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

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

— Nondeterministic Choice

lemma *R-choice*: $(rel-R \lceil P \rceil \lceil Q \rceil) \cup (rel-R \lceil P \rceil \lceil Q \rceil) \leq rel-R \lceil P \rceil \lceil Q \rceil$
using *rel-rkat.R-choice*[of $\lceil P \rceil \lceil Q \rceil$].

lemma *R-choice-law*: $X \leq rel-R \lceil P \rceil \lceil Q \rceil \implies Y \leq rel-R \lceil P \rceil \lceil Q \rceil \implies X \cup Y \leq rel-R \lceil P \rceil \lceil Q \rceil$
using *le-supI*.

lemma *R-choice-mono*: $P' \subseteq P \implies Q' \subseteq Q \implies P' \cup Q' \subseteq P \cup Q$
using *Un-mono*.

— Assignment

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

lemma *R-assign-law*:
 $(\forall s. P s \longrightarrow Q (\chi j. (((\$) s)(x := (e s))) j)) \implies (x ::= e) \leq rel-R \lceil P \rceil \lceil Q \rceil$
unfolding *sH-assign[symmetric]* **by** (rule *rel-rkat.R2*)

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

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

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

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

— Nondeterministic Assignment

lemma *R-nondet-assign*: $(x ::= ?) \leq rel-R \lceil \lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j) \rceil \lceil P \rceil$
unfolding *rel-rkat.spec-def* **by** (rule *H-nondet-assign*)

lemma *R-nondet-assign-law*:
 $(\forall s. P s \longrightarrow (\forall k. Q (\chi j. (((\$) s)(x := k)) j))) \implies (x ::= ?) \leq rel-R \lceil P \rceil \lceil Q \rceil$
unfolding *sH-nondet-assign[symmetric]* **by** (rule *rel-rkat.R2*)

— Conditional Statement

lemma *R-cond*:

$(IF\ B\ THEN\ rel-R\ [\lambda s.\ B\ s \wedge P\ s]\ [Q]\ ELSE\ rel-R\ [\lambda s.\ \neg\ B\ s \wedge P\ s]\ [Q]) \leq rel-R\ [P]\ [Q]$
using *rel-rkat.R-cond*[of $[B]\ [P]\ [Q]$] **by** *simp*

lemma *R-cond-mono*: $X \leq X' \implies Y \leq Y' \implies (IF\ P\ THEN\ X\ ELSE\ Y) \leq IF\ P\ THEN\ X'\ ELSE\ Y'$
by (*auto simp: rel-kat.kat-cond-def*)

lemma *R-cond-law*: $X \leq rel-R\ [\lambda s.\ B\ s \wedge P\ s]\ [Q] \implies Y \leq rel-R\ [\lambda s.\ \neg\ B\ s \wedge P\ s]\ [Q] \implies (IF\ B\ THEN\ X\ ELSE\ Y) \leq rel-R\ [P]\ [Q]$
by (*rule order-trans; (rule R-cond-mono)?, (rule R-cond)? auto*)

— While Loop

lemma *R-while*: $K = (\lambda s.\ P\ s \wedge \neg\ B\ s) \implies WHILE\ B\ INV\ I\ DO\ (rel-R\ [\lambda s.\ P\ s \wedge B\ s]\ [P]) \leq rel-R\ [P]\ [K]$
unfolding *rel-kat.kat-while-inv-def* **using** *rel-rkat.R-while*[of $[B]\ [P]$] **by** *simp*

lemma *R-whileI*:

$X \leq rel-R\ [I]\ [I] \implies [P] \leq [\lambda s.\ I\ s \wedge B\ s] \implies [\lambda s.\ I\ s \wedge \neg\ B\ s] \leq [Q] \implies WHILE\ B\ INV\ I\ DO\ X \leq rel-R\ [P]\ [Q]$
by (*rule rel-rkat.R2, rule rel-kat.H-while-inv, auto simp: rel-kat-H rel-rkat.spec-def*)

lemma *R-while-mono*: $X \leq X' \implies (WHILE\ P\ INV\ I\ DO\ X) \subseteq WHILE\ P\ INV\ I\ DO\ X'$
by (*simp add: rel-diod.mult-isol rel-diod.mult-isor rel-ka.conway.dagger-iso rel-kat.kat-while-def rel-kat.kat-while-inv-def*)

lemma *R-while-law*: $X \leq rel-R\ [\lambda s.\ P\ s \wedge B\ s]\ [P] \implies Q = (\lambda s.\ P\ s \wedge \neg\ B\ s) \implies (WHILE\ B\ INV\ I\ DO\ X) \leq rel-R\ [P]\ [Q]$
by (*rule order-trans; (rule R-while-mono)?, (rule R-while)?*)

— Finite Iteration

lemma *R-loop*: $LOOP\ rel-R\ [P]\ [P]\ INV\ I \leq rel-R\ [P]\ [P]$
using *rel-rkat.R-loop* .

lemma *R-loopI*:

$X \leq rel-R\ [I]\ [I] \implies [P] \leq [I] \implies [I] \leq [Q] \implies LOOP\ X\ INV\ I \leq rel-R\ [P]\ [Q]$
unfolding *rel-rkat.spec-def* **using** *H-loopI* **by** *blast*

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

— Evolution command (flow)

lemma *R-g-evol*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(EVOL \varphi G U) \leq rel-R [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow P (\varphi t s)] [P]$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-evol, simp*)

lemma *R-g-evol-law*:

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

lemma *R-g-evoll*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \Longrightarrow (EVOL \varphi G U) ; rel-R [R] [Q] \leq rel-R [P] [Q]$
apply(*rule-tac R=R in R-seq-law*)
by (*rule-tac R-g-evol-law, simp-all*)

lemma *R-g-evolr*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \Longrightarrow rel-R [P] [R]; (EVOL \varphi G U) \leq rel-R [P] [Q]$
apply(*rule-tac R=R in R-seq-law, simp*)
by (*rule-tac R-g-evol-law, simp*)

lemma

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $EVOL \varphi G U ; rel-R [Q] [Q] \leq rel-R [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)] [Q]$
by (*rule R-g-evoll simp*)

lemma

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $rel-R [Q] [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)]; EVOL \varphi G U \leq rel-R [Q] [Q]$
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 $(x' = (\lambda t. f) \ \& \ G \ on \ U \ S \ @ \ 0) \leq rel-R [\lambda s. s \in S \longrightarrow (\forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow P (\varphi t s))] [P]$

unfolding *rel-rkat.spec-def* **by** (*rule H-g-ode-subset[OF assms]*, *simp-all*)

lemma *R-g-ode-rule-subset*:

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

lemma *R-g-odel-subset*:

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

lemma *R-g-oder-subset*:

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

lemma *R-g-ode*: $(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) S \ @ \ 0) \leq$

$\text{rel-R } [\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow P (\varphi t s))] \ [P]$
by (*rule R-g-ode-subset*, *auto simp: init-time interval-time*)

lemma *R-g-ode-law*: $(\forall s \in S. P s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$

$(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
unfolding *sH-g-ode[symmetric]* **by** (*rule rel-rkat.R2*)

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \implies$

$(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) S \ @ \ 0) ; \text{rel-R } [R] \ [Q] \leq \text{rel-R } [P] \ [Q]$
by (*rule R-g-odel-subset*, *auto simp: init-time interval-time*)

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \implies$

$\text{rel-R } [P] \ [R] ; (x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
by (*rule R-g-oder-subset*, *auto simp: init-time interval-time*)

lemma *R-g-ode-ivl*:

$t \geq 0 \implies t \in T \implies (\forall s \in S. P s \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$

$(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{0..t\}) S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
unfolding *sH-g-ode-ivl[symmetric]* **by** (*rule rel-rkat.R2*)

end

— Evolution command (invariants)

lemma *R-g-ode-inv: diff-invariant I f T S t₀ G* $\implies [P] \leq [I] \implies [\lambda s. I s \wedge G s] \leq [Q] \implies$
 $(x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0 \ \text{DINV} \ I) \leq \text{rel-R} \ [P] \ [Q]$
unfolding *rel-rkat.spec-def* **by** (*auto simp: H-g-ode-inv*)

7.5 Derivation of the rules of dL

We derive rules of differential dynamic logic (dL). This allows the components to reason in the style of that logic.

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

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

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 $\{P\} \ (x' = f \ \& \ G) \ \{Q\}$
using *assms* **by** (*subst local-flow.sH-g-ode-subset, auto*)

lemma *diff-solve-rule2*:

fixes $c::\text{'a}::\{\text{heine-borel}, \text{banach}\}$
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 $\{P\} \ (x' = (\lambda s. c) \ \& \ G) \ \{Q\}$
apply (*subst local-flow.sH-g-ode-subset* [**where** $T = \text{UNIV}$ **and** $\varphi = (\lambda t \ x. x + t *_R c)$])
using *line-is-local-flow assms* **by** *auto*

lemma *diff-weak-rule*:

assumes $[G] \leq [Q]$
shows $\{P\} \ (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{Q\}$
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* $[P] \ (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [C]$
and *wp-Q:rel-kat.Hoare* $[P] \ (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ [Q]$
shows $\{P\} \ (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{Q\}$
proof (*subst rel-kat-H, simp add: g-orbital-eq p2r-def g-ode-def, clarsimp*)
fix $t::\text{real}$ **and** $X::\text{real} \Rightarrow \text{'a}$ **and** s
assume $P \ s$ **and** $t \in U \ s$
and $x\text{-ivp}: X \in \text{ivp-sols} \ f \ U \ S \ t_0 \ s$
and *guard-x*: $\forall x. x \in U \ s \wedge x \leq t \longrightarrow G \ (X \ x)$
have $\forall t \in (\text{down} \ (U \ s) \ t). X \ t \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
using *g-orbitalI[OF x-ivp] guard-x* **by** *auto*
hence $\forall t \in (\text{down} \ (U \ s) \ t). C \ (X \ t)$

```

    using wp-C ⟨P s⟩ by (subst (asm) rel-kat-H, auto simp: g-ode-def)
  hence X t ∈ g-orbital f (λs. G s ∧ C s) U S t₀ 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) rel-kat-H) (auto simp: g-ode-def)
qed

```

lemma *diff-inv-rule*:

```

  assumes [P] ≤ [I] and diff-invariant I f U S t₀ G and [I] ≤ [Q]
  shows {P} (x' = f & G on U S @ t₀) {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

7.6 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
    HOL-Eisbach.Eisbach

```

begin

— A tactic for verification of hybrid programs

named-theorems *hoare-intros*

```

  declare H-assignI [hoare-intros]
    and H-cond [hoare-intros]
    and local-flow.H-g-ode-subset [hoare-intros]
    and H-g-ode-inv [hoare-intros]

```

method *body-hoare*

```

  = (rule hoare-intros,(simp)?; body-hoare?)

```

method *hyb-hoare* **for** *P::'a pred*

```

  = (rule H-loopI, rule H-seq[where R=P]; body-hoare?)

```

— A tactic for refinement of hybrid programs

named-theorems *refine-intros selected refinement lemmas*

```

  declare R-loopI [refine-intros]
    and R-loop-mono [refine-intros]
    and R-cond-law [refine-intros]
    and R-cond-mono [refine-intros]

```

and *R-while-law* [*refine-intros*]
and *R-assignl* [*refine-intros*]
and *R-seq-law* [*refine-intros*]
and *R-seq-mono* [*refine-intros*]
and *R-g-evol-law* [*refine-intros*]
and *R-skip* [*refine-intros*]
and *R-g-ode-inv* [*refine-intros*]

method *refinement*
 = (*rule refine-intros; (refinement)?*)

7.6.1 Pendulum

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

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

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

— Verified with annotated dynamics

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

— Verified with differential invariants

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

— Verified with the flow

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

no-notation $fpend$ (f)
and $pend-flow$ (φ)

7.6.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

abbreviation $fball :: real \Rightarrow real^2 \Rightarrow real^2$ (f)
where $f g s \equiv (\chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } g)$

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

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

qed

lemma $fball-invariant$:

fixes $g h :: real$
defines $dinv: I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows $diff-invariant$ I ($\lambda t. f g$) ($\lambda s. UNIV$) $UNIV$ 0 G
unfolding $dinv$ **apply**($rule$ $diff-invariant-rules$, $simp$)
by($auto$ $intro!$: $poly-derivatives$)

lemma $bouncing-ball-inv: g < 0 \implies h \geq 0 \implies$

```

{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(hyb-hoare λs::real^2. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
using fball-invariant 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 · τ2 / 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 · τ2 + 2 · v · τ + 2 · x = 0 by auto
then have g2 · τ2 + 2 · g · v · τ + 2 · g · x = 0
by (metis (mono-tags) Groups.mult-ac(1,3) mult-zero-right
  monoid-mult-class.power2-eq-square semiring-class.distrib-left)
hence g2 · τ2 + 2 · g · v · τ + v2 + 2 · g · h = 0
using invar by (simp add: monoid-mult-class.power2-eq-square)
hence obs: (g · τ + v)2 + 2 · g · h = 0
apply(subst power2-sum) by (metis (no-types) Groups.add-ac(2, 3)
  Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2))
thus 2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0
by (simp add: monoid-mult-class.power2-eq-square)
have 2 · g · h + (- ((g · τ) + v))2 = 0
using obs by (metis Groups.add-ac(2) power2-minus)
thus 2 · g · h + (- (g · τ) - v) · (- (g · τ) - v) = 0
by (simp add: monoid-mult-class.power2-eq-square)

```

qed

lemma [bb-real-arith]:

```

assumes invar: 2 · g · x = 2 · g · h + v · v
shows 2 · g · (g · τ2 / 2 + v · τ + (x::real)) =
  2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (is ?lhs = ?rhs)

```

proof –

```

have ?lhs = g2 · τ2 + 2 · g · v · τ + 2 · g · x
by(auto simp: algebra-simps semiring-normalization-rules(29))
also have ... = g2 · τ2 + 2 · g · v · τ + 2 · g · h + v · v (is ... = ?middle)
by(subst invar, simp)
finally have ?lhs = ?middle.

```

moreover

```

{have ?rhs = g · g · (τ · τ) + 2 · g · v · τ + 2 · g · h + v · v
by (simp add: Groups.mult-ac(2,3) semiring-class.distrib-left)
also have ... = ?middle

```

by (*simp add: semiring-normalization-rules(29)*)
finally have $?rhs = ?middle.$
ultimately show $?thesis$ **by** *auto*
qed

lemma *bouncing-ball-dyn*: $g < 0 \implies h \geq 0 \implies$
 $\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
((EVOL (φ g) ($\lambda s. s\$1 \geq 0$) T);
(IF ($\lambda s. s\$1 = 0$) THEN ($2 ::= (\lambda s. - s\$2)$) ELSE skip))
INV ($\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
apply(*hyb-hoare* $\lambda s::\text{real}^2. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
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*: $g < 0 \implies h \geq 0 \implies$
 $\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
(($x' = f$ g & ($\lambda s. s\$1 \geq 0$));
(IF ($\lambda s. s\$1 = 0$) THEN ($2 ::= (\lambda s. - s\$2)$) ELSE skip))
INV ($\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
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])?)
apply(*subst local-flow.sH-g-ode-subset[OF local-flow-ball]*)
by (*auto simp: bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g < (0::\text{real}) \implies 0 \leq h \implies$
 $2 ::= (\lambda s. - s\$2) \leq \text{rel-}R$
 $[\lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
 $[\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
by (*rule R-assign-law, auto*)

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows *rel-R* $[\lambda s. s\$1 = h \wedge s\$2 = 0]$ $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h] \geq$
(LOOP
((EVOL (φ 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(refinement; (rule R-bb-assign[OF assms])?)
using assms by (auto simp: bb-real-arith)

```

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

7.6.3 Thermostat

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

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

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

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

abbreviation *therm-flow* :: $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$ (φ)
where φ a *L* τ *s* $\equiv (\chi i. \text{if } i = 1 \text{ then } -\exp(-a * \tau) * (L - s\$1) + L \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \implies \|f \text{ a } L \ s_1 - f \text{ a } L \ s_2\| = |a| * |s_1\$1 - s_2\$1|$

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*: $a > 0 \implies$ *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*: $0 < T_{min} T_{min} \leq T T \leq T_{max}$
and *thyps*: $0 \leq (\tau::real) \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(T_{min} / T) / a)$
shows $T_{min} \leq \exp(-a * \tau) * T$ **and** $\exp(-a * \tau) * T \leq T_{max}$

proof –

have $0 \leq \tau \wedge \tau \leq -(\ln(T_{min} / T) / a)$
using *thyps by auto*
hence $\ln(T_{min} / T) \leq -a * \tau \wedge -a * \tau \leq 0$
using *assms(1) divide-le-cancel by fastforce*
also have $T_{min} / T > 0$
using *Thyps by auto*
ultimately have *obs*: $T_{min} / T \leq \exp(-a * \tau) \exp(-a * \tau) \leq 1$
using *exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less, simp)*
thus $T_{min} \leq \exp(-a * \tau) * T$
using *Thyps by (simp add: pos-divide-le-eq)*
show $\exp(-a * \tau) * T \leq T_{max}$
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*: $T_{min} \leq T T \leq T_{max} T_{max} < (L::real)$
and *thyps*: $0 \leq \tau \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$
shows $L - T_{max} \leq \exp(-(a * \tau)) * (L - T)$
and $L - \exp(-(a * \tau)) * (L - T) \leq T_{max}$
and $T_{min} \leq L - \exp(-(a * \tau)) * (L - T)$

proof –

have $0 \leq \tau \wedge \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$
using *thyps by auto*
hence $\ln((L - T_{max}) / (L - T)) \leq -a * \tau \wedge -a * \tau \leq 0$
using *assms(1) divide-le-cancel by fastforce*
also have $(L - T_{max}) / (L - T) > 0$
using *Thyps by auto*
ultimately have $(L - T_{max}) / (L - T) \leq \exp(-a * \tau) \wedge \exp(-a * \tau) \leq 1$

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 $\{I\ Tmin\ Tmax\}$

(*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*

$(x' = (\lambda t. f\ a\ 0) \ \&\ G\ Tmin\ Tmax\ a\ 0\ \text{on}\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0)$

ELSE

$(x' = (\lambda t. f\ a\ L) \ \&\ G\ Tmin\ Tmax\ a\ L\ \text{on}\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0)$)

) *INV* $I\ Tmin\ Tmax$)

$\{I\ Tmin\ Tmax\}$

apply(*rule H-loopI*)

apply(*rule-tac* $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0$ **in** *H-seq, simp*)

apply(*rule-tac* $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$1 = s\3 **in** *H-seq, simp*)

apply(*rule-tac* $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$1 = s\3 **in** *H-seq, simp*)

apply(*rule H-cond, simp-all add: H-g-ode-therm[OF assms(1,2)], safe*)

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 *rel-R* $[\lambda s. s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] [I\ Tmin\ Tmax] \geq$

$(x' = (\lambda t. f a 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on} \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-down-real-arith[OF assms(1,3), of - Tmax] **by** auto

lemma R-therm-dyn-up:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $rel\text{-}R \ [\lambda s. s\$4 \neq 0 \wedge I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$
 $(x' = (\lambda t. f a L) \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on} \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin] **by** auto

lemma R-therm-dyn:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $rel\text{-}R \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$
 $(IF \ (\lambda s. s\$4 = 0) \ THEN$
 $(x' = (\lambda t. f a 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on} \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
 $ELSE$
 $(x' = (\lambda t. f a L) \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on} \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0))$
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: $rel\text{-}R \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \geq$
 $(2 ::= (\lambda s. 0))$
by (auto simp: R-assign-law)

lemma R-therm-assign2:

$rel\text{-}R \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$
 $(3 ::= (\lambda s. s\$1))$
by (auto simp: R-assign-law)

lemma R-therm-ctrl:

$rel\text{-}R \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$
 $(2 ::= (\lambda s. 0));$
 $(3 ::= (\lambda s. s\$1));$
 $(IF \ (\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1) \ THEN$
 $(4 ::= (\lambda s. 1))$
 $ELSE \ IF \ (\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1) \ THEN$
 $(4 ::= (\lambda s. 0))$
 $ELSE \ skip)$
apply(refinement, rule R-therm-assign1, rule R-therm-assign2)
by (rule R-assign-law, simp)+ auto

lemma R-therm-loop: $rel\text{-}R \ [I \ Tmin \ Tmax] \ [I \ Tmin \ Tmax] \geq$
 $(LOOP$

$rel\text{-}R \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1];$

$rel\text{-}R \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax]$
 $INV \ I \ Tmin \ Tmax)$
by (*intro R-loopI R-seq, simp-all*)

lemma *R-thermostat-flow*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows $rel\text{-}R \ [I \ Tmin \ Tmax] \ [I \ Tmin \ Tmax] \geq$

$(LOOP \ ($
 — 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
 $(IF \ (\lambda s. s\$4 = 0) \ THEN$
 $(x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
 $ELSE$
 $(x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0))$
 $) \ INV \ I \ Tmin \ Tmax)$
apply(*refinement*; (*rule R-therm-assign1*)?, (*rule R-therm-assign2*)?,
 (*rule R-therm-dyn-down*)?, (*rule R-therm-dyn-up*)?, (*rule R-assign-law*)?)
using *assms* **by** *auto*

no-notation *therm-vec-field* (f)

and *therm-flow* (φ)
and *therm-guard* (G)
and *therm-loop-inv* (I)

7.6.4 Water tank

7.6.5 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $hmin \leq h \leq hmax$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

abbreviation *tank-vec-field* $:: real \Rightarrow real^4 \Rightarrow real^4 \ (f)$

where $f \ k \ s \equiv (\chi \ i. \text{if } i = 2 \text{ then } 1 \text{ else } (if \ i = 1 \text{ then } k \text{ else } 0))$

abbreviation *tank-flow* $:: real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 \ (\varphi)$

where $\varphi \ k \ \tau \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } k * \tau + s\$1 \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

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

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

abbreviation $\text{tank-diff-inv} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{bool} \ (dI)$
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 $\text{local-flow-tank}: \text{local-flow} \ (f \ k) \ \text{UNIV} \ \text{UNIV} \ (\varphi \ k)$

apply ($\text{unfold-locates}, \text{unfold local-lipschitz-def lipschitz-on-def}, \text{simp-all}, \text{clar-simp}$)

apply($\text{rule-tac } x=1/2 \ \text{in } exI, \text{clarsimp}, \text{rule-tac } x=1 \ \text{in } exI$)

apply($\text{simp add: dist-norm norm-vec-def L2-set-def}, \text{unfold UNIV-4}$)

by ($\text{auto intro!: poly-derivatives simp: vec-eq-iff}$)

lemma tank-arith :

assumes $0 \leq (\tau::\text{real})$ **and** $0 < c_o$ **and** $c_o < c_i$

shows $\forall \tau \in \{0..\tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$

and $\forall \tau \in \{0..\tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$

and $hmin \leq y \implies hmin \leq (c_i - c_o) \cdot \tau + y$

and $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$

apply($\text{simp-all add: field-simps le-divide-eq assms}$)

using $\text{assms apply} \ (\text{meson add-mono less-eq-real-def mult-left-mono})$

using $\text{assms by} \ (\text{meson add-increasing2 less-eq-real-def mult-nonneg-nonneg})$

lemmas $H\text{-g-ode-tank} = \text{local-flow.sh-g-ode-ivl}[OF \ \text{local-flow-tank} - \text{UNIV-I}]$

lemma tank-flow :

assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$

shows $\{I \ hmin \ hmax\}$

(LOOP

— control

$((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$

$(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) \ \text{THEN } (4 ::= (\lambda s. 1)) \ \text{ELSE}$

$(\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \ \text{THEN } (4 ::= (\lambda s. 0)) \ \text{ELSE skip});$

— dynamics

$(\text{IF } (\lambda s. s\$4 = 0) \ \text{THEN } (x' = (\lambda t. f \ (c_i - c_o))) \ \& \ G \ hmax \ (c_i - c_o) \ \text{on } (\lambda s.$

$\{0..\tau\}) \ \text{UNIV} \ @ \ 0)$

$\ \text{ELSE } (x' = (\lambda t. f \ (-c_o))) \ \& \ G \ hmin \ (-c_o) \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0)) \)$

$\text{INV } I \ hmin \ hmax)$

$\{I \ hmin \ hmax\}$

apply($\text{rule } H\text{-loopI}$)

```

apply(rule-tac R= $\lambda s. I \text{ hmin hmax } s \wedge s\$2=0$  in H-seq, simp)
apply(rule-tac R= $\lambda s. I \text{ hmin hmax } s \wedge s\$2=0 \wedge s\$3 = s\$1$  in H-seq, simp)
apply(rule-tac R= $\lambda s. I \text{ hmin hmax } s \wedge s\$2=0 \wedge s\$3 = s\$1$  in H-seq, 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:

```

 $0 \leq \tau \implies \text{diff-invariant } (dI \text{ hmin hmax } k) (\lambda t. f k) (\lambda s. \{0..\tau\}) \text{ UNIV } 0 \text{ Guard}$ 
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  $0 \leq (\tau::\text{real})$  and  $c_o < c_i$  and  $b: \text{ hmin} \leq y_0$  and  $g: \tau \leq (\text{hmax} - y_0)$ 
/ $(c_i - c_o)$ 
shows  $\text{ hmin} \leq (c_i - c_o) \cdot \tau + y_0$  and  $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$ 
proof -
have  $(c_i - c_o) \cdot \tau \leq (\text{hmax} - y_0)$ 
using g assms(2,3) by (metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq)
thus  $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$ 
by auto
show  $\text{ hmin} \leq (c_i - c_o) \cdot \tau + y_0$ 
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  $0 \leq (\tau::\text{real})$  and  $0 < c_o$  and  $b: y_0 \leq \text{hmax}$  and  $g: \tau \leq -((\text{hmin} - y_0) / c_o)$ 
shows  $\text{ hmin} \leq y_0 - c_o \cdot \tau$  and  $y_0 - c_o \cdot \tau \leq \text{hmax}$ 
proof -
have  $\tau \cdot c_o \leq y_0 - \text{hmin}$ 
using g  $\langle 0 < c_o \rangle$  pos-le-minus-divide-eq by fastforce
thus  $\text{ hmin} \leq y_0 - c_o \cdot \tau$ 
by (auto simp: mult.commute)
show  $y_0 - c_o \cdot \tau \leq \text{hmax}$ 
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  $\{I \text{ hmin hmax}\}$ 
(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
 $(IF (\lambda s. s\$4 = 0) THEN$
 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ hmax (c_i - c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV$
 $(dI \ hmin \ hmax (c_i - c_o)))$
 $ELSE$
 $(x' = (\lambda t. f (-c_o)) \ \& \ G \ hmin (-c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV (dI$
 $hmin \ hmax (-c_o)))$)
 $INV \ I \ hmin \ hmax$
 $\{I \ hmin \ hmax\}$
apply(rule *H-loopI*)
apply(rule-tac $R = \lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0$ in *H-seq, simp*)
apply(rule-tac $R = \lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\1 in *H-seq, simp*)
apply(rule-tac $R = \lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\1 in *H-seq, simp*)
apply(rule *H-cond, simp*)
apply(rule *H-g-ode-inv, simp add: tank-diff-inv[OF assms(1)], clarsimp, clarsimp*)
using *assms tank-inv-arith1* **apply** *force*
apply(rule *H-g-ode-inv, simp only: sH-diff-inv*)
apply(rule *tank-diff-inv[OF assms(1)]*)
using *assms tank-inv-arith2* **by** *auto*

— Refined with differential invariants

abbreviation *tank-ctrl* :: $real \Rightarrow real \Rightarrow (real^4) \ rel \ (ctrl)$
where $ctrl \ hmin \ hmax \equiv ((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) THEN (4 ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip))$

abbreviation *tank-dyn-dinv* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow (real^4) \ rel$
 (dyn)
where $dyn \ c_i \ c_o \ hmin \ hmax \ \tau \equiv (IF (\lambda s. s\$4 = 0) THEN$
 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ hmax (c_i - c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV$
 $(dI \ hmin \ hmax (c_i - c_o)))$
 $ELSE$
 $(x' = (\lambda t. f (-c_o)) \ \& \ G \ hmin (-c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV (dI$
 $hmin \ hmax (-c_o)))$

abbreviation *tank-dinv* $c_i \ c_o \ hmin \ hmax \ \tau \equiv$
 $LOOP (ctrl \ hmin \ hmax ; dyn \ c_i \ c_o \ hmin \ hmax \ \tau) \ INV (I \ hmin \ hmax)$

lemma *R-tank-inv*:

assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $rel\text{-}R \ [I \ hmin \ hmax] \ [I \ hmin \ hmax] \ \geq$
 $(LOOP$
— control


```

((2 ::= (λs. 0)); (3 ::= (λs. s$1)));
(IF (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) THEN (4 ::= (λs. 1)) ELSE
(IF (λs. s$4 = 1 ∧ s$3 ≥ hmax - 1) THEN (4 ::= (λs. 0)) ELSE skip));
— dynamics
(IF (λs. s$4 = 0) THEN
(x' = (λt. f (ci - co)) & G hmax (ci - co) on (λs. {0..τ}) UNIV @ 0 DINV
(dI hmin hmax (ci - co)))
ELSE
(x' = (λt. f (-co)) & G hmin (-co) on (λs. {0..τ}) UNIV @ 0 DINV (dI
hmin hmax (-co))) )
INV I hmin hmax)
proof —
have rel-R [I hmin hmax] [I hmin hmax] ≥
LOOP (
(2 ::= (λs. 0)); (rel-R [λs. I hmin hmax s ∧ s$2 = 0] [I hmin hmax])
) INV I hmin hmax (is - ≥ ?R)
by (refinement, auto)
moreover have
?R ≥ LOOP (
(2 ::= (λs. 0)); (3 ::= (λs. s$1));
(rel-R [λs. I hmin hmax s ∧ s$2 = 0 ∧ s$3 = s$1] [I hmin hmax])
) INV I hmin hmax (is - ≥ ?R)
by (refinement, auto)
moreover have
?R ≥ LOOP (
ctrl hmin hmax;
(rel-R [λs. I hmin hmax s ∧ s$2 = 0 ∧ s$3 = s$1] [I hmin hmax])
) INV I hmin hmax (is - ≥ ?R)
by (simp only: O-assoc, refinement; (force)?, (rule R-assign-law)?) auto
moreover have
?R ≥ LOOP (ctrl hmin hmax; dyn ci co hmin hmax τ) INV I hmin hmax
apply (simp only: O-assoc, refinement; ((rule tank-diff-inv[OF assms(1)])? |
(simp)?)
using tank-inv-arith1 tank-inv-arith2 assms by auto
ultimately show ?thesis
by (auto simp: O-assoc)
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

```

7.7 Verification of hybrid programs

We use our state transformers model to obtain verification and refinement components for hybrid programs. We retake the three methods for reasoning with evolution commands and their continuous dynamics: providing flows, solutions or invariants.

```

theory HS-VC-KAT-ndfun
  imports
    ../HS-ODEs
    HS-VC-KAT
    ../HS-VC-KA-ndfun

begin

instantiation nd-fun :: (type) kat
begin

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

lemma nd-fun-n-op-one[nd-fun-ka]:  $n\ (n\ (1 :: 'a\ \text{nd-fun})) = 1$ 
  and nd-fun-n-op-mult[nd-fun-ka]:  $n\ (n\ (n\ x \cdot n\ y)) = n\ x \cdot n\ y$ 
  and nd-fun-n-op-mult-comp[nd-fun-ka]:  $n\ x \cdot n\ (n\ x) = 0$ 
  and nd-fun-n-op-de-morgan[nd-fun-ka]:  $n\ (n\ (n\ x) \cdot n\ (n\ y)) = n\ x + n\ y$  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-ka)

end

instantiation nd-fun :: (type) rkat
begin

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

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

```

7.7.1 Regular programs

Lemmas for manipulation of predicates in the relational model

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

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)

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

lemma $p2ndf\text{-simps}[simp]$:

$\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P s \longrightarrow Q s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P s = Q s)$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P s \wedge Q s \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P s \vee Q s \rceil$
 $\text{tt } \lceil P \rceil = \lceil P \rceil$
 $n \lceil P \rceil = \lceil \lambda s. \neg P s \rceil$

unfolding $p2ndf\text{-def one-nd-fun-def less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def}$

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

Lemmas for verification condition generation

abbreviation $ndfunHoare (\{ _ \} - \{ _ \})$
where $\{P\} X \{Q\} \equiv Hoare \lceil P \rceil X \lceil Q \rceil$

lemma $ndfun\text{-kat-H}: \{P\} X \{Q\} \longleftrightarrow (\forall s s'. P s \longrightarrow s' \in (X \bullet) s \longrightarrow Q s')$
unfolding $Hoare\text{-def } p2ndf\text{-def less-eq-nd-fun-def times-nd-fun-def kcomp\text{-def}$
by (*auto simp add: le-fun-def n-op-nd-fun-def*)

— Skip

abbreviation $skip \equiv (1 :: 'a \text{ nd-fun})$

lemma $sH\text{-skip}[simp]: \{P\} skip \{Q\} \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
unfolding $ndfun\text{-kat-H}$ **by** (*simp add: one-nd-fun-def*)

lemma $H\text{-skip}: \{P\} skip \{P\}$
by *simp*

— Tests

lemma $sH\text{-test}[simp]: \{P\} \lceil R \rceil \{Q\} = (\forall s. P s \longrightarrow R s \longrightarrow Q s)$
by (*subst ndfun-kat-H, simp add: p2ndf-def*)

— Abort

abbreviation $abort \equiv (0 :: 'a \text{ nd-fun})$

lemma *sH-abort*[*simp*]: $\{P\} \text{ abort } \{Q\} \longleftrightarrow \text{True}$
unfolding *ndfun-kat-H* **by** (*simp add: zero-nd-fun-def*)

lemma *H-abort*: $\{P\} \text{ abort } \{Q\}$
by *simp*

— Assignments

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

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

lemma *H-assign*: $P = (\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j)) \Longrightarrow \{P\} (x ::= e) \{Q\}$
by *simp*

— Nondeterministic assignments

definition *nondet-assign* :: $'b \Rightarrow ('a \hat{\sim} b) \text{ nd-fun } ((_ ::= _) [70] 61)$
where $(x ::= ?) = (\lambda s. \{\text{vec-upd } s \ x \ k \mid k. \text{True}\})^\bullet$

lemma *sH-nondet-assign*[*simp*]:
 $\{P\} (x ::= ?) \{Q\} \longleftrightarrow (\forall s. P \ s \longrightarrow (\forall k. Q \ (\chi \ j. (((\$) \ s)(x := k)) \ j)))$
unfolding *ndfun-kat-H vec-upd-def nondet-assign-def* **by** (*auto simp: fun-upd-def*)

lemma *H-nondet-assign*: $\{\lambda s. \forall k. P \ (\chi \ j. (((\$) \ s)(x := k)) \ j)\} (x ::= ?) \{P\}$
by *simp*

— Sequential Composition

abbreviation *seq-seq* :: $'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun}$ (**infixl** ; 75)
where $f ; g \equiv f \cdot g$

lemma *H-seq*: $\{P\} X \{R\} \Longrightarrow \{R\} Y \{Q\} \Longrightarrow \{P\} X;Y \{Q\}$
by (*auto intro: H-seq*)

lemma *sH-seq*: $\{P\} X;Y \{Q\} = \{P\} X \{\lambda s. \forall s'. s' \in (Y \bullet) \ s \longrightarrow Q \ s'\}$
unfolding *ndfun-kat-H* **by** (*auto simp: times-nd-fun-def kcomp-def*)

lemma *H-assignl*:
assumes $\{K\} X \{Q\}$
and $\forall s. P \ s \longrightarrow K \ (\text{vec-lambda } (((\$) \ s)(x := e \ s)))$
shows $\{P\} (x ::= e);X \{Q\}$
apply(*rule H-seq, subst sH-assign*)
using *assms* **by** *auto*

— Nondeterministic Choice

lemma *sH-choice*: $\{P\} X + Y \{Q\} \longleftrightarrow (\{P\} X \{Q\} \wedge \{P\} Y \{Q\})$
unfolding *ndfun-kat-H* **by** (*auto simp: plus-nd-fun-def*)

lemma *H-choice*: $\{P\} X \{Q\} \Longrightarrow \{P\} Y \{Q\} \Longrightarrow \{P\} X + Y \{Q\}$
using *H-choice* .

— Conditional Statement

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

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

lemma *H-cond*:
 $\{\lambda s. P s \wedge B s\} X \{Q\} \Longrightarrow \{\lambda s. P s \wedge \neg B s\} Y \{Q\} \Longrightarrow \{P\} (\text{IF } B \text{ THEN } X \text{ ELSE } Y) \{Q\}$
by *simp*

— While Loop

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

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

lemma $\{\lambda s. P s \wedge B s\} X \{\lambda s. P s\} \Longrightarrow \{P\} (\text{WHILE } B \text{ INV } I \text{ DO } X) \{\lambda s. P s \wedge \neg B s\}$
using *H-while[of [P] [B] X]*
unfolding *kat-while-inv-def* **by** *auto*

— Finite Iteration

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

lemma *H-loop*: $\{P\} X \{P\} \Longrightarrow \{P\} (\text{LOOP } X \text{ INV } I) \{P\}$
by (*auto intro: H-loop*)

lemma *H-loopI*: $\{I\} X \{I\} \Longrightarrow [P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow \{P\} (\text{LOOP } X$

INV I) {*Q*}
using *H-loop-inv*[of [*P*] [*I*] *X* [*Q*]] **by** *auto*

7.7.2 Evolution commands

definition *g-evol* :: (('a::ord) ⇒ 'b ⇒ 'b) ⇒ 'b pred ⇒ ('b ⇒ 'a set) ⇒ 'b nd-fun
(*EVOL*)

where *EVOL* φ *G* *U* = ($\lambda s. g\text{-orbit } (\lambda t. \varphi t s) G (U s)$)[•]

lemma *sH-g-evol[simp]*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$

shows {*P*} (*EVOL* φ *G* *U*) {*Q*} = ($\forall s. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))$)

unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** *auto*

lemma *H-g-evol*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$

assumes $P = (\lambda s. (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

shows {*P*} (*EVOL* φ *G* *U*) {*Q*}

by (*simp add: assms*)

— Verification by providing solutions

definition *g-ode* :: (real ⇒ ('a::banach) ⇒ 'a) ⇒ 'a pred ⇒ ('a ⇒ real set) ⇒ 'a set
⇒

real ⇒ 'a nd-fun (($1x' = - \& - \text{on } - - @ -$))

where ($x' = f \& G \text{ on } U S @ t_0$) ≡ ($\lambda s. g\text{-orbital } f G U S t_0 s$)[•]

lemma *H-g-orbital*:

$P = (\lambda s. (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))) \implies$

{*P*} ($x' = f \& G \text{ on } U S @ t_0$) {*Q*}

unfolding *ndfun-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

lemma *sH-g-orbital*: {*P*} ($x' = f \& G \text{ on } U S @ t_0$) {*Q*} =

($\forall s. P s \longrightarrow (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))$)

unfolding *g-orbital-eq g-ode-def ndfun-kat-H* **by** *auto*

context *local-flow*

begin

lemma *sH-g-ode-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

shows {*P*} ($x' = (\lambda t. f) \& G \text{ on } U S @ 0$) {*Q*} =

($\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))$)

proof(*unfold sH-g-orbital, clarsimp, safe*)

fix *s t*

assume *hyp*s: $s \in S P s t \in U s \forall \tau. \tau \in U s \wedge \tau \leq t \longrightarrow G (\varphi \tau s)$

and main: $\forall s. P s \longrightarrow (\forall X \in \text{Sols } (\lambda t. f) U S 0 s. \forall t \in U s. (\forall \tau. \tau \in U s \wedge \tau \leq t \longrightarrow G (X \tau)) \longrightarrow Q (X t))$
hence $(\lambda t. \varphi t s) \in \text{Sols } (\lambda t. f) U S 0 s$
using *in-ivp-sols assms by blast*
thus $Q (\varphi t s)$
using *main hyps by fastforce*
next
fix $s X t$
assume *hyps:* $P s X \in \text{Sols } (\lambda t. f) U S 0 s t \in U s \forall \tau. \tau \in U s \wedge \tau \leq t \longrightarrow G (X \tau)$
and main: $\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau. \tau \in U s \wedge \tau \leq t \longrightarrow G (\varphi \tau s)) \longrightarrow Q (\varphi t s))$
hence *obs:* $s \in S$
using *ivp-sols-def[of $\lambda t. f$] init-time by auto*
hence $\forall \tau \in \text{down } (U s) t. X \tau = \varphi \tau s$
using *eq-solution hyps assms by blast*
thus $Q (X t)$
using *hyps main obs by auto*
qed

lemma *H-g-ode-subset:*

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

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

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

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

end

— Verification with differential invariants

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

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

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

lemma *H-g-ode-inv*: $\{I\} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \{I\} \implies \lceil P \rceil \leq \lceil I \rceil \implies \lceil \lambda s. I s \wedge G s \rceil \leq \lceil Q \rceil \implies \{P\} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \text{DINV } I \{Q\}$
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*)

7.8 Refinement Components

lemma *R-skip*: $(\forall s. P s \longrightarrow Q s) \implies 1 \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
by (*auto simp: spec-def ndfun-kat-H one-nd-fun-def*)

— Abort

lemma *R-abort*: $\text{abort} \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
by (*rule R2, simp*)

— Sequential Composition

lemma *R-seq*: $(\text{Ref } \lceil P \rceil \lceil R \rceil) ; (\text{Ref } \lceil R \rceil \lceil Q \rceil) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
using *R-seq* **by** *blast*

lemma *R-seq-law*: $X \leq \text{Ref } \lceil P \rceil \lceil R \rceil \implies Y \leq \text{Ref } \lceil R \rceil \lceil Q \rceil \implies X ; Y \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
unfolding *spec-def* **by** (*rule H-seq*)

lemmas *R-seq-mono = mult-isol-var*

— Nondeterministic Choice

lemma *R-choice*: $(\text{Ref } \lceil P \rceil \lceil Q \rceil) + (\text{Ref } \lceil P \rceil \lceil Q \rceil) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
using *R-choice[of $\lceil P \rceil \lceil Q \rceil$]* .

lemma *R-choice-law*: $X \leq \text{Ref } [P] [Q] \implies Y \leq \text{Ref } [P] [Q] \implies X + Y \leq \text{Ref } [P] [Q]$
using *join.le-supI* .

lemma *R-choice-mono*: $P' \leq P \implies Q' \leq Q \implies P' + Q' \leq P + Q$
using *set-plus-mono2* .

— Assignment

lemma *R-assign*: $(x ::= e) \leq \text{Ref } [\lambda s. P (\chi j. (((\$) s)(x := e s)) j)] [P]$
unfolding *spec-def* **by** (*rule H-assign, clarsimp simp: fun-eq-iff fun-upd-def*)

lemma *R-assign-law*:
 $(\forall s. P s \longrightarrow Q (\chi j. (((\$) s)(x := (e s))) j)) \implies (x ::= e) \leq \text{Ref } [P] [Q]$
unfolding *sH-assign[symmetric] spec-def* .

lemma *R-assignl*:
 $P = (\lambda s. R (\chi j. (((\$) s)(x := e s)) j)) \implies (x ::= e) ; \text{Ref } [R] [Q] \leq \text{Ref } [P] [Q]$
apply(*rule-tac R=R in R-seq-law*)
by (*rule-tac R-assign-law, simp-all*)

lemma *R-assignr*:
 $R = (\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)) \implies \text{Ref } [P] [R]; (x ::= e) \leq \text{Ref } [P] [Q]$
apply(*rule-tac R=R in R-seq-law, simp*)
by (*rule-tac R-assign-law, simp*)

lemma $(x ::= e) ; \text{Ref } [Q] [Q] \leq \text{Ref } [(\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)] [Q]$
by (*rule R-assignl simp*)

lemma $\text{Ref } [Q] [(\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)]; (x ::= e) \leq \text{Ref } [Q] [Q]$
by (*rule R-assignr simp*)

— Nondeterministic Assignment

lemma *R-nondet-assign*: $(x ::= ?) \leq \text{Ref } [\lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j)] [P]$
unfolding *spec-def* **by** (*rule H-nondet-assign*)

lemma *R-nondet-assign-law*:
 $(\forall s. P s \longrightarrow (\forall k. Q (\chi j. (((\$) s)(x := k)) j))) \implies (x ::= ?) \leq \text{Ref } [P] [Q]$
unfolding *sH-nondet-assign[symmetric] by (rule R2)*

— Conditional Statement

lemma *R-cond*:
 $(\text{IF } B \text{ THEN } \text{Ref } [\lambda s. B s \wedge P s] [Q] \text{ ELSE } \text{Ref } [\lambda s. \neg B s \wedge P s] [Q]) \leq \text{Ref } [Q] [Q]$

$[P] [Q]$
using *R-cond*[of $[B] [P] [Q]$] **by** *simp*

lemma *R-cond-mono*: $X \leq X' \implies Y \leq Y' \implies (IF P THEN X ELSE Y) \leq IF P THEN X' ELSE Y'$

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*)

lemma *R-cond-law*: $X \leq Ref [\lambda s. B s \wedge P s] [Q] \implies Y \leq Ref [\lambda s. \neg B s \wedge P s] [Q] \implies$

$(IF B THEN X ELSE Y) \leq Ref [P] [Q]$

by (*rule order-trans; (rule R-cond-mono)?, (rule R-cond)? auto*)

— While loop

lemma *R-while*: $K = (\lambda s. P s \wedge \neg B s) \implies$

$WHILE B INV I DO (Ref [\lambda s. P s \wedge B s] [P]) \leq Ref [P] [K]$

unfolding *kat-while-inv-def* **using** *R-while*[of $[B] [P]$] **by** *simp*

lemma *R-whileI*:

$X \leq Ref [I] [I] \implies [P] \leq [\lambda s. I s \wedge B s] \implies [\lambda s. I s \wedge \neg B s] \leq [Q] \implies$

$WHILE B INV I DO X \leq Ref [P] [Q]$

by (*rule R2, rule H-while-inv, auto simp: ndfun-kat-H spec-def*)

lemma *R-while-mono*: $X \leq X' \implies (WHILE P INV I DO X) \leq WHILE P INV I DO X'$

by (*simp add: kat-while-inv-def kat-while-def mult-isol mult-isor star-iso*)

lemma *R-while-law*: $X \leq Ref [\lambda s. P s \wedge B s] [P] \implies Q = (\lambda s. P s \wedge \neg B s) \implies$

$(WHILE B INV I DO X) \leq Ref [P] [Q]$

by (*rule order-trans; (rule R-while-mono)?, (rule R-while)?*)

— Finite Iteration

lemma *R-loop*: $X \leq Ref [I] [I] \implies [P] \leq [I] \implies [I] \leq [Q] \implies LOOP X INV I \leq Ref [P] [Q]$

unfolding *spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loopI*: $X \leq Ref [I] [I] \implies [P] \leq [I] \implies [I] \leq [Q] \implies LOOP X INV I \leq Ref [P] [Q]$

unfolding *spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \leq X' \implies LOOP X INV I \leq LOOP X' INV I$

unfolding *kat-loop-inv-def* **by** (*simp add: star-iso*)

— Evolution command (flow)

lemma *R-g-evol*:

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

lemma *R-g-evol-law*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall s. P s \longrightarrow (\forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$
 $(EVOL \varphi G U) \leq Ref [P] [Q]$
unfolding sH-g-evol[symmetric] spec-def .

lemma *R-g-evoll*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \implies$
 $(EVOL \varphi G U) ; Ref [R] [Q] \leq Ref [P] [Q]$
apply(rule-tac *R=R* in *R-seq-law*)
by (rule-tac *R-g-evol-law*, *simp-all*)

lemma *R-g-evolr*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \implies$
 $Ref [P] [R]; (EVOL \varphi G U) \leq Ref [P] [Q]$
apply(rule-tac *R=R* in *R-seq-law*, *simp*)
by (rule-tac *R-g-evol-law*, *simp*)

lemma

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $EVOL \varphi G U ; Ref [Q] [Q] \leq$
 $Ref [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)] [Q]$
by (rule *R-g-evoll*) *simp*

lemma

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

— Evolution command (ode)

context *local-flow*

begin

lemma *R-g-ode-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge is\text{-interval } (U s) \wedge U s \subseteq T$
shows $(x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) \leq$
 $Ref [\lambda s. s \in S \longrightarrow (\forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow P (\varphi t s))] [P]$
unfolding spec-def by (rule *H-g-ode-subset[OF assms]*, *auto*)

lemma *R-g-ode-rule-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

shows $(\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$

\implies

$(x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

unfolding *spec-def* **by** (*subst sH-g-ode-subset[OF assms], auto*)

lemma *R-g-odel-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

and $P = (\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s))$

shows $(x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) ; \text{Ref } [R] \ [Q] \leq \text{Ref } [P] \ [Q]$

apply (*rule-tac R=R in R-seq-law, rule-tac R-g-ode-rule-subset*)

by (*simp-all add: assms*)

lemma *R-g-oder-subset*:

assumes $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$

and $R = (\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))$

shows $\text{Ref } [P] \ [R]; (x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

apply (*rule-tac R=R in R-seq-law, simp*)

by (*rule-tac R-g-ode-rule-subset, simp-all add: assms*)

lemma *R-g-ode*: $(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) \leq$

$\text{Ref } [\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow P (\varphi t s))] \ [P]$

by (*rule R-g-ode-subset, auto simp: init-time interval-time*)

lemma *R-g-ode-law*: $(\forall s \in S. P s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$

$(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

unfolding *sH-g-ode[symmetric]* **by** (*rule R2*)

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \implies$

$(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) ; \text{Ref } [R] \ [Q] \leq \text{Ref } [P] \ [Q]$

by (*rule R-g-odel-subset, auto simp: init-time interval-time*)

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \implies$

$\text{Ref } [P] \ [R]; (x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

by (*rule R-g-oder-subset, auto simp: init-time interval-time*)

lemma *R-g-ode-ivl*:

$t \geq 0 \implies t \in T \implies (\forall s \in S. P s \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$

$(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. \{0..t\}) S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

unfolding *sH-g-ode-ivl[symmetric]* **by** (*rule R2*)

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $I f T S t_0 G \implies [P] \leq [I] \implies [\lambda s. I s \wedge G$

$s] \leq [Q] \implies$
 $(x'=f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0 \ \text{DINV} \ I) \leq \text{Ref} \ [P] \ [Q]$
unfolding spec-def by (*auto simp: H-g-ode-inv*)

7.9 Derivation of the rules of dL

We derive rules of differential dynamic logic (dL). This allows the components to reason in the style of that logic.

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

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

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 $\{P\} \ (x'=f \ \& \ G) \ \{Q\}$
using *assms by(subst local-flow.sH-g-ode-subset, auto)*

lemma *diff-solve-rule2*:

fixes $c::a::\{\text{heine-borel}, \text{banach}\}$
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 $\{P\} \ (x'=(\lambda s. c) \ \& \ G) \ \{Q\}$
apply(*subst local-flow.sH-g-ode-subset[where T=UNIV and $\varphi=(\lambda t \ x. x + t *_R c)$*])
using *line-is-local-flow assms by auto*

lemma *diff-weak-rule*:

assumes $[G] \leq [Q]$
shows $\{P\} \ (x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{Q\}$
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 $[P] \ (x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ [C]$*
and *wp-Q:Hoare $[P] \ (x'=f \ \& \ (\lambda s. G \ s \ \wedge \ C \ s) \ \text{on} \ U \ S \ @ \ t_0) \ [Q]$*
shows $\{P\} \ (x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{Q\}$
proof(*subst ndfun-kat-H, simp add: g-orbital-eq p2ndf-def g-ode-def, clarsimp*)
fix $t::\text{real}$ **and** $X::\text{real} \Rightarrow a$ **and** s
assume $P \ s$ **and** $t \in U \ s$
and $x\text{-ivp}:X \in \text{ivp-sols} \ f \ U \ S \ t_0 \ s$
and *guard-x:* $\forall x. x \in U \ s \ \wedge \ x \leq t \longrightarrow G \ (X \ x)$
have $\forall t \in (\text{down} \ (U \ s) \ t). X \ t \in \text{g-orbital} \ f \ G \ U \ S \ t_0 \ s$
using *g-orbitalI[OF x-ivp] guard-x by auto*
hence $\forall t \in (\text{down} \ (U \ s) \ t). C \ (X \ t)$
using *wp-C $\langle P \ s \rangle$ by (subst (asm) ndfun-kat-H, auto simp: g-ode-def)*
hence $X \ t \in \text{g-orbital} \ f \ (\lambda s. G \ s \ \wedge \ C \ s) \ U \ S \ t_0 \ s$

```

    using guard-x ⟨t ∈ U s⟩ by (auto intro!: g-orbitaI 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 t0 G and [I] ≤ [Q]
  shows {P} (x' = f & G on U S @ t0) {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

7.10 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
    HOL-Eisbach.Eisbach

```

begin

— A tactic for verification of hybrid programs

named-theorems *hoare-intros*

```

  declare H-assignl [hoare-intros]
    and H-cond [hoare-intros]
    and local-flow.H-g-ode-subset [hoare-intros]
    and H-g-ode-inv [hoare-intros]

```

method *body-hoare*

```

  = (rule hoare-intros,(simp)?; body-hoare?)

```

method *hyb-hoare* **for** *P::'a pred*

```

  = (rule H-loopI, rule H-seq[where R=P]; body-hoare?)

```

— A tactic for refinement of hybrid programs

named-theorems *refine-intros selected refinement lemmas*

```

  declare R-loopI [refine-intros]
    and R-loop-mono [refine-intros]
    and R-cond-law [refine-intros]
    and R-cond-mono [refine-intros]
    and R-while-law [refine-intros]
    and R-assignl [refine-intros]

```

and *R-seq-law* [*refine-intros*]
and *R-seq-mono* [*refine-intros*]
and *R-g-evol-law* [*refine-intros*]
and *R-skip* [*refine-intros*]
and *R-g-ode-inv* [*refine-intros*]

method *refinement*
 = (*rule refine-intros; (refinement)?*)

7.10.1 Pendulum

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

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

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

— Verified with annotated dynamics

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

— Verified with differential invariants

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

no-notation *fpend* (*f*)

and *pend-flow* (φ)

7.10.2 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. That is, we model it as a completely elastic collision with the ground. We use $s\$1$ to represent the ball's height and $s\$2$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

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

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

— 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** $\langle 0 > g \rangle$ **by** *auto*

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

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

hence $(v \cdot v) / (2 \cdot g) = (x - h)$

by *auto*

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

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

ultimately have $h - x \geq 0$

by *linarith*

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

qed

lemma *fball-invariant*:

fixes $g h :: real$

defines *dinv*: $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$

shows *diff-invariant* I ($\lambda t. f g$) ($\lambda s. UNIV$) $UNIV$ $0 G$

unfolding *dinv* **apply**(*rule diff-invariant-rules, simp*)

by(*auto intro!*: *poly-derivatives*)

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies$

$\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$

(*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(hyb-hoare λs::real^2. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2)
using fball-invariant 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 · τ2 / 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 · τ2 + 2 · v · τ + 2 · x = 0 by auto
then have g2 · τ2 + 2 · g · v · τ + 2 · g · x = 0
by (metis (mono-tags) Groups.mult-ac(1,3) mult-zero-right
monoid-mult-class.power2-eq-square semiring-class.distrib-left)
hence g2 · τ2 + 2 · g · v · τ + v2 + 2 · g · h = 0
using invar by (simp add: monoid-mult-class.power2-eq-square)
hence obs: (g · τ + v)2 + 2 · g · h = 0
apply(subst power2-sum) by (metis (no-types) Groups.add-ac(2, 3)
Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2))
thus 2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0
by (simp add: monoid-mult-class.power2-eq-square)
have 2 · g · h + (- ((g · τ) + v))2 = 0
using obs by (metis Groups.add-ac(2) power2-minus)
thus 2 · g · h + (- (g · τ) - v) · (- (g · τ) - v) = 0
by (simp add: monoid-mult-class.power2-eq-square)

```

qed

lemma [bb-real-arith]:

```

assumes invar: 2 · g · x = 2 · g · h + v · v
shows 2 · g · (g · τ2 / 2 + v · τ + (x::real)) =
2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (is ?lhs = ?rhs)

```

proof—

```

have ?lhs = g2 · τ2 + 2 · g · v · τ + 2 · g · x
by(auto simp: algebra-simps semiring-normalization-rules(29))
also have ... = g2 · τ2 + 2 · g · v · τ + 2 · g · h + v · v (is ... = ?middle)
by(subst invar, simp)
finally have ?lhs = ?middle.

```

moreover

```

{have ?rhs = g · g · (τ · τ) + 2 · g · v · τ + 2 · g · h + v · v
by (simp add: Groups.mult-ac(2,3) semiring-class.distrib-left)
also have ... = ?middle
by (simp add: semiring-normalization-rules(29))
finally have ?rhs = ?middle.}

```

ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*: $g < 0 \implies h \geq 0 \implies$
 $\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
((EVOL $(\varphi\ g)$ $(\lambda\ s. s\$1 \geq 0)$ T);
(IF $(\lambda\ s. s\$1 = 0)$ *THEN* $(\lambda s. - s\$2)$ *ELSE skip*))
INV $(\lambda s. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
apply(*hyb-hoare* $\lambda s::\text{real}^2. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
by (*auto simp: bb-real-arith*)

— Verified with the flow

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

lemma *bouncing-ball-flow*: $g < 0 \implies h \geq 0 \implies$
 $\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
((x' = f g & (\lambda s. s\\$1 \geq 0));
(IF $(\lambda\ s. s\$1 = 0)$ *THEN* $(\lambda s. - s\$2)$ *ELSE skip*))
INV $(\lambda s. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
apply(*rule H-loopI; (rule H-seq[where R= $\lambda s. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$])?*)
apply(*subst local-flow.sH-g-ode-subset[OF local-flow-ball]*)
by (*auto simp: bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g < (0::\text{real}) \implies 0 \leq h \implies$
 $\lambda s. - s\$2 \leq \text{Ref}$
 $\lceil \lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
 $\lceil \lambda s. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
by (*rule R-assign-law, 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* $(\lambda s. - s\$2)$ *ELSE skip*))
INV $(\lambda s. 0 \leq s\$1 \wedge \lambda s. 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$)

apply(*refinement*; (*rule R-bb-assign*[*OF assms*])?)
using *assms* **by** (*auto simp: bb-real-arith*)

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

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

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

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

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

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \implies \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$

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-lsqr*) *auto*

lemma *local-flow-therm*: $a > 0 \implies$ *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*: $0 < T_{min} T_{min} \leq T T \leq T_{max}$
and *thyps*: $0 \leq (\tau::real) \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(T_{min} / T) / a)$
shows $T_{min} \leq \exp(-a * \tau) * T$ **and** $\exp(-a * \tau) * T \leq T_{max}$

proof –

have $0 \leq \tau \wedge \tau \leq -(\ln(T_{min} / T) / a)$
using *thyps by auto*
hence $\ln(T_{min} / T) \leq -a * \tau \wedge -a * \tau \leq 0$
using *assms(1) divide-le-cancel by fastforce*
also have $T_{min} / T > 0$
using *Thyps by auto*
ultimately have *obs*: $T_{min} / T \leq \exp(-a * \tau) \exp(-a * \tau) \leq 1$
using *exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less, simp)*
thus $T_{min} \leq \exp(-a * \tau) * T$
using *Thyps by (simp add: pos-divide-le-eq)*
show $\exp(-a * \tau) * T \leq T_{max}$
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*: $T_{min} \leq T T \leq T_{max} T_{max} < (L::real)$
and *thyps*: $0 \leq \tau \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$
shows $L - T_{max} \leq \exp(-(a * \tau)) * (L - T)$
and $L - \exp(-(a * \tau)) * (L - T) \leq T_{max}$
and $T_{min} \leq L - \exp(-(a * \tau)) * (L - T)$

proof –

have $0 \leq \tau \wedge \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$
using *thyps by auto*
hence $\ln((L - T_{max}) / (L - T)) \leq -a * \tau \wedge -a * \tau \leq 0$
using *assms(1) divide-le-cancel by fastforce*
also have $(L - T_{max}) / (L - T) > 0$
using *Thyps by auto*
ultimately have $(L - T_{max}) / (L - T) \leq \exp(-a * \tau) \wedge \exp(-a * \tau) \leq 1$
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 $\{I\ Tmin\ Tmax\}$

(*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 ($\lambda s. \{0..\tau\}$) *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*)

$\{I\ Tmin\ Tmax\}$

apply(*rule H-loopI*)

apply(*rule-tac R= $\lambda s. I\ Tmin\ Tmax\ s \wedge s\$2=0 \wedge s\$3 = s\1 in H-seq*)

apply(*rule-tac R= $\lambda s. I\ Tmin\ Tmax\ s \wedge s\$2=0 \wedge s\$3 = s\1 in H-seq*)

apply(*rule-tac R= $\lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0$ in H-seq, simp, simp*)

apply(*rule H-cond, simp-all add: H-g-ode-therm[OF assms(1,2)]*)+

using *therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin]*

and *therm-dyn-down-real-arith[OF assms(1,3), of - Tmax]* **by** *auto*

— Refined with the flow

lemma *R-therm-dyn-down*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows *Ref* [$\lambda s. s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I Tmin Tmax*] \geq

($x' = (\lambda t. f\ a\ 0) \ \&\ G\ Tmin\ Tmax\ a\ 0$ on ($\lambda s. \{0..\tau\}$) *UNIV @ 0*)

apply(*rule local-flow.R-g-ode-ivl[OF local-flow-therm]*)

using *assms therm-dyn-down-real-arith*[*OF assms(1,3)*, *of - Tmax*] **by** *auto*

lemma *R-therm-dyn-up*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows *Ref* [$\lambda s. s\$4 \neq 0 \wedge I Tmin Tmax s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I Tmin Tmax*] \geq

($x' = (\lambda t. f a L) \ \& \ G \ Tmin \ Tmax \ a \ L$ on $(\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$)

apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])

using *assms therm-dyn-up-real-arith*[*OF assms(1) - - assms(4)*, *of Tmin*] **by** *auto*

lemma *R-therm-dyn*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows *Ref* [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I Tmin Tmax*] \geq

(*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 $(\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$))

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*] [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0$] \geq
($2 ::= (\lambda s. 0)$)

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

lemma *R-therm-assign2*:

Ref [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0$] [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0 \wedge s\$3 = s\1] \geq ($3 ::= (\lambda s. s\$1)$)

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

lemma *R-therm-ctrl*:

Ref [*I Tmin Tmax*] [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0 \wedge s\$3 = s\1] \geq

($2 ::= (\lambda s. 0)$);

($3 ::= (\lambda s. s\$1)$);

(*IF* ($\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1$) *THEN*

($4 ::= (\lambda s. 1)$)

ELSE IF ($\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$) *THEN*

($4 ::= (\lambda s. 0)$)

ELSE skip)

apply(*refinement*, *rule R-therm-assign1*, *rule R-therm-assign2*)

by (*rule R-assign-law*, *simp*) + *auto*

lemma *R-therm-loop*: *Ref* [*I Tmin Tmax*] [*I Tmin Tmax*] \geq
(*LOOP*

Ref [*I Tmin Tmax*] [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0 \wedge s\$3 = s\1];

Ref [$\lambda s. I Tmin Tmax s \wedge s\$2 = 0 \wedge 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** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$

shows $Ref [I T_{min} T_{max}] [I T_{min} T_{max}] \geq$

(*LOOP* (

— control

($2 ::= (\lambda s. 0)$);($3 ::= (\lambda s. s\$1)$);

(*IF* ($\lambda s. s\$4 = 0 \wedge s\$3 \leq T_{min} + 1$) *THEN*

($4 ::= (\lambda s. 1)$)

ELSE IF ($\lambda s. s\$4 = 1 \wedge s\$3 \geq T_{max} - 1$) *THEN*

($4 ::= (\lambda s. 0)$)

ELSE skip);

— dynamics

(*IF* ($\lambda s. s\$4 = 0$) *THEN*

($x' = (\lambda t. f a 0)$ & $G T_{min} T_{max} a 0$ on ($\lambda s. \{0.. \tau\}$) *UNIV @ 0*)

ELSE

($x' = (\lambda t. f a L)$ & $G T_{min} T_{max} a L$ on ($\lambda s. \{0.. \tau\}$) *UNIV @ 0*))

) *INV I T_{min} T_{max}*)

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)

7.10.4 Water tank

7.10.5 Tank

A controller turns a water pump on and off to keep the level of water h in a tank within an acceptable range $h_{min} \leq h \leq h_{max}$. Just like in the previous example, after each intervention, the controller registers the current level of water and resets its chronometer, then it changes the status of the water pump accordingly. The level of water grows linearly $h' = k$ at a rate of $k = c_i - c_o$ if the pump is on, and at a rate of $k = -c_o$ if the pump is off. We use 1 to denote the tank's level of water, 2 is time as measured by the controller's chronometer, 3 is the level of water measured by the chronometer, and 4 states whether the pump is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the controller keeps the level of water between h_{min} and h_{max} .

abbreviation *tank-vec-field* $:: real \Rightarrow real^4 \Rightarrow real^4 (f)$

where $f k s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } k \text{ else } 0))$

abbreviation *tank-flow* $:: real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (\varphi)$

where $\varphi k \tau s \equiv (\chi i. \text{if } i = 1 \text{ then } k * \tau + s\$1 \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

abbreviation *tank-guard* $:: real \Rightarrow real \Rightarrow real^4 \Rightarrow bool (G)$

where $G \ Hm \ k \ s \equiv s\$2 \leq (Hm - s\$3)/k$

abbreviation *tank-loop-inv* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool \ (I)$

where $I \ hmin \ hmax \ s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

abbreviation *tank-diff-inv* :: $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow bool \ (dI)$

where $dI \ hmin \ hmax \ k \ s \equiv s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge$

$hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

— Verified with the flow

lemma *local-flow-tank*: *local-flow* ($f \ k$) *UNIV UNIV* ($\varphi \ k$)

apply (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clar-simp*)

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 $\forall \tau \in \{0.. \tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$

and $\forall \tau \in \{0.. \tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$

and $hmin \leq y \implies hmin \leq (c_i - c_o) \cdot \tau + y$

and $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$

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 $\{I \ hmin \ hmax\}$

(*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

$(IF (\lambda s. s\$4 = 0) THEN (x' = (\lambda t. f (c_i - c_o))) \& G \ hmax (c_i - c_o) \text{ on } (\lambda s.$

$\{0.. \tau\}) \text{ UNIV @ } 0)$

$ELSE (x' = (\lambda t. f (-c_o))) \& G \ hmin (-c_o) \text{ on } (\lambda s. \{0.. \tau\}) \text{ UNIV @ } 0)))$

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*:

$0 \leq \tau \implies \text{diff-invariant } (dI \text{ hmin hmax } k) (\lambda t. f \ k) (\lambda s. \{0..\tau\}) \text{ UNIV } 0 \text{ Guard}$

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 $0 \leq (\tau::\text{real})$ **and** $c_o < c_i$ **and** $b: \text{hmin} \leq y_0$ **and** $g: \tau \leq (\text{hmax} - y_0) / (c_i - c_o)$

shows $\text{hmin} \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$

proof –

have $(c_i - c_o) \cdot \tau \leq (\text{hmax} - y_0)$

using *g assms(2,3)* **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)

thus $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$

by *auto*

show $\text{hmin} \leq (c_i - c_o) \cdot \tau + y_0$

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 $0 \leq (\tau::\text{real})$ **and** $0 < c_o$ **and** $b: y_0 \leq \text{hmax}$ **and** $g: \tau \leq -((\text{hmin} - y_0) / c_o)$

shows $\text{hmin} \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq \text{hmax}$

proof –

have $\tau \cdot c_o \leq y_0 - \text{hmin}$

using *g $\langle 0 < c_o \rangle$ pos-le-minus-divide-eq* **by** *fastforce*

thus $\text{hmin} \leq y_0 - c_o \cdot \tau$

by (*auto simp: mult.commute*)

show $y_0 - c_o \cdot \tau \leq \text{hmax}$

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 $\{I \text{ hmin hmax}\}$

(*LOOP*

— *control*

(*(2 ::= ($\lambda s. 0$)); (3 ::= ($\lambda s. s\$1$))*);

(*IF ($\lambda s. s\$4 = 0 \wedge s\$3 \leq \text{hmin} + 1$) THEN ($4 ::= (\lambda s. 1)$) ELSE*

(*IF ($\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{hmax} - 1$) 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 \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, 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

abbreviation tank-ctrl :: $real \Rightarrow real \Rightarrow (real^4)$ nd-fun (ctrl)
 where ctrl hmin hmax $\equiv ((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 (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)))

abbreviation tank-dyn-dinv :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow (real^4)$
 nd-fun (dyn)
 where dyn $c_i \ c_o \ hmin \ hmax \ \tau \equiv (IF (\lambda s. s\$4 = 0) THEN$
 ($x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ hmax (c_i - c_o)$ 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 ($-c_o$)))

abbreviation tank-dinv $c_i \ c_o \ hmin \ hmax \ \tau \equiv LOOP (ctrl \ hmin \ hmax ; dyn \ c_i \ c_o$
 $hmin \ hmax \ \tau) \ INV (I \ hmin \ hmax)$

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] \geq

(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 DINV

```

(dI hmin hmax (ci-co))
  ELSE
    (x' = (λt. f (-co)) & G hmin (-co) on (λs. {0..τ}) UNIV @ 0 DINV (dI
hmin hmax (-co))) )
  INV I hmin hmax)
proof -
  have Ref [I hmin hmax] [I hmin hmax] ≥
    LOOP (
      (? ::= (λs. 0)); (Ref [λs. I hmin hmax s ∧ s$2 = 0] [I hmin hmax])
    ) INV I hmin hmax (is - ≥ ?R)
  by (refinement, auto)
  moreover have
    ?R ≥ LOOP (
      (? ::= (λs. 0)); (? ::= (λs. s$1));
      (Ref [λs. I hmin hmax s ∧ s$2 = 0 ∧ s$3=s$1] [I hmin hmax])
    ) INV I hmin hmax (is - ≥ ?R)
  by (simp only: mult.assoc, refinement, auto)
  moreover have
    ?R ≥ LOOP (
      ctrl hmin hmax;
      (Ref [λs. I hmin hmax s ∧ s$2 = 0 ∧ s$3=s$1] [I hmin hmax])
    ) INV I hmin hmax (is - ≥ ?R)
  by (simp only: mult.assoc, refinement; (force)?, (rule R-assign-law)?) auto
  moreover have
    ?R ≥ LOOP (ctrl hmin hmax; dyn ci co hmin hmax τ) INV I hmin hmax
  apply(simp only: mult.assoc, refinement; ((rule tank-diff-inv[OF assms(1)])) ? |
(simp)?)
  using tank-inv-arith1 tank-inv-arith2 assms by auto
  ultimately show ?thesis
  by auto
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

References

- [1] A. Armstrong, V. B. F. Gomes, and G. Struth. Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing*, 28(2):265–293, 2016.
- [2] B. Bohrer, V. Rahli, I. Vukotic, M. Völpl, and A. Platzer. Formally verified differential dynamic logic. In *CPP 2017*, pages 208–221. ACM,

2017.

- [3] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.
- [4] V. B. F. Gomes and G. Struth. Program construction and verification components based on Kleene algebra. *Archive of Formal Proofs*, 2016.
- [5] F. Immler and J. Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, 2012.
- [6] A. Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.
- [7] G. Struth. Transformer semantics. *Archive of Formal Proofs*, 2018.