

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	5
2.3	Intermediate Value Theorem	7
2.4	Filters	7
2.5	Multivariable derivatives	8
3	Ordinary Differential Equations	9
3.1	Initial value problems and orbits	10
3.2	Differential Invariants	11
3.3	Picard-Lindelöf	13
3.4	Flows for ODEs	15
4	Verification components for hybrid systems	17
4.1	Verification of regular programs	18
4.2	Verification of hybrid programs	20
4.3	Derivation of the rules of dL	22
4.4	Examples	24
4.4.1	Pendulum	24
4.4.2	Bouncing Ball	25

4.4.3	Thermostat	26
4.4.4	Tank	28
5	Verification components with Predicate Transformers	29
5.1	Verification of regular programs	29
5.2	Verification of hybrid programs	31
5.3	Derivation of the rules of dL	33
5.4	Examples	35
5.4.1	Pendulum	35
5.4.2	Bouncing Ball	36
5.4.3	Thermostat	37
5.4.4	Tank	39
6	Verification components with MKA	40
6.1	Verification in AKA	40
6.2	Relational model	43
6.3	Verification of hybrid programs	43
6.3.1	Regular programs	44
6.3.2	Evolution commands	45
6.3.3	Derivation of the rules of dL	47
6.4	Examples	49
6.4.1	Pendulum	49
6.4.2	Bouncing Ball	50
6.4.3	Thermostat	51
6.4.4	Tank	53
6.5	State transformer model	54
6.6	Verification of hybrid programs	56
6.6.1	Regular programs	56
6.6.2	Evolution commands	58
6.6.3	Derivation of the rules of dL	60
6.7	Examples	61
6.7.1	Pendulum	62
6.7.2	Bouncing Ball	62
6.7.3	Thermostat	64
6.7.4	Tank	65
7	Verification components with KAT	67
7.1	Hoare logic in KAT	67
7.2	refinement KAT	69
7.3	Verification of hybrid programs	70
7.3.1	Regular programs	71
7.3.2	Evolution commands	73
7.4	Refinement Components	75
7.5	Derivation of the rules of dL	79

7.6	Examples	80
7.6.1	Pendulum	81
7.6.2	Bouncing Ball	82
7.6.3	Thermostat	83
7.6.4	Water tank	87
7.6.5	Tank	87
7.7	Verification of hybrid programs	89
7.7.1	Regular programs	90
7.7.2	Evolution commands	93
7.8	Refinement Components	95
7.9	Derivation of the rules of dL	99
7.10	Examples	100
7.10.1	Pendulum	101
7.10.2	Bouncing Ball	101
7.10.3	Thermostat	103
7.10.4	Water tank	106
7.10.5	Tank	106

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-eg*:

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

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

<proof>

lemma *real-ivl-egs*:

assumes $0 < r$

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

lemma *is-interval-real-nonneg*[simp]: *is-interval* (Collect ((\leq) (0::real)))
 $\langle \text{proof} \rangle$

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

lemma *sum-eq-Sum*:
assumes *inj-on* $f \ A$
shows $(\sum_{x \in A} f \ x) = (\sum \{f \ x \mid x. x \in A\})$
 $\langle \text{proof} \rangle$

lemma *triangle-norm-vec-le-sum*: $\|x\| \leq (\sum_{i \in UNIV} \|x \ \$ \ i\|)$
 $\langle \text{proof} \rangle$

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 $D \ f = f' \ \text{on } g \ ' \ T$
and $D \ g = g' \ \text{on } T$
and $h = (\lambda t. g' \ t *_{\mathbb{R}} f' \ (g \ t))$
shows $D \ (\lambda t. f \ (g \ t)) = h \ \text{on } T$
 $\langle \text{proof} \rangle$

lemma *vderiv-uminusI*[poly-derivatives]:
 $D \ f = f' \ \text{on } T \implies g = (\lambda t. - f' \ t) \implies D \ (\lambda t. - f \ t) = g \ \text{on } T$
 $\langle \text{proof} \rangle$

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

fixes $f::real \Rightarrow real$

assumes $n \geq 1$ **and** $D f = f'$ on T **and** $g = (\lambda t. n * (f' t) * (f t)^{\wedge}(n-1))$

shows $D (\lambda t. (f t)^{\wedge}n) = g$ on T

<proof>

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

assumes $\forall t \in T. g t \neq (0::real)$ **and** $D f = f'$ on T **and** $D g = g'$ on T

and $h = (\lambda t. (f' t * g t - f t * (g' t)) / (g t)^{\wedge}2)$

shows $D (\lambda t. (f t)/(g t)) = h$ on T

<proof>

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

assumes $D (f::real \Rightarrow real) = f'$ on T **and** $g = (\lambda t. - (f' t) * \sin (f t))$

shows $D (\lambda t. \cos (f t)) = g$ on T

<proof>

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

<proof>

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

<proof>

lemma $D (*) a = (\lambda t. a)$ on T

<proof>

lemma $a \neq 0 \implies D (\lambda t. t/a) = (\lambda t. 1/a)$ on T

<proof>

lemma $(a::real) \neq 0 \implies D f = f'$ on $T \implies g = (\lambda t. (f' t)/a) \implies D (\lambda t. (f t)/a) = g$ on T

<proof>

lemma $\forall t \in T. f t \neq (0::real) \implies D f = f'$ on $T \implies g = (\lambda t. - a * f' t / (f t)^{\wedge}2) \implies$

$D (\lambda t. a/(f t)) = g$ on T

<proof>

lemma $D (\lambda t. a * t^2 / 2 + v * t + x) = (\lambda t. a * t + v)$ on T

<proof>

lemma $D (\lambda t. v * t - a * t^2 / 2 + x) = (\lambda x. v - a * x)$ on T

<proof>

lemma $D x = x'$ on $(\lambda \tau. \tau + t) ' T \implies D (\lambda \tau. x (\tau + t)) = (\lambda \tau. x' (\tau + t))$ on

T
<proof>

lemma $a \neq 0 \implies D (\lambda t. t/a) = (\lambda t. 1/a)$ on *T*
<proof>

lemma $c \neq 0 \implies D (\lambda t. a5 * t^5 + a3 * (t^3 / c) - a2 * \exp (t^2) + a1 * \cos t + a0) =$
 $(\lambda t. 5 * a5 * t^4 + 3 * a3 * (t^2 / c) - 2 * a2 * t * \exp (t^2) - a1 * \sin t)$
on *T*
<proof>

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*
<proof>

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*
<proof>

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>

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>

2.4 Filters

lemma *eventually-at-within-mono*:
assumes $t \in interior\ T$ **and** $T \subseteq S$
and *eventually* *P* (at *t* within *T*)
shows *eventually* *P* (at *t* within *S*)
<proof>

lemma *netlimit-at-within-mono*:
fixes $t::'a::\{perfect-space, t2-space\}$
assumes $t \in interior\ T$ **and** $T \subseteq S$
shows *netlimit* (at *t* within *S*) = *t*

<proof>

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

<proof>

lemma *eventually-all-finite2*:

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

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

shows $\text{eventually } (\lambda x. \forall i. P i x) F$

<proof>

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 $\text{eventually } Q F$

<proof>

2.5 Multivariable derivatives

definition $\text{vec-upd} :: ('a \sim b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a \sim b$

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

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

<proof>

lemma *has-derivative-vec-nth*[*derivative-intros*]:

$D (\lambda s. s \$ i) \mapsto (\lambda s. s \$ i)$ (at x within S)

<proof>

lemma *bounded-linear-component*:

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

<proof>

lemma *open-preimage-nth*:

$\text{open } S \implies \text{open } \{s :: ('a::\text{real-normed-vector} \sim n::\text{finite}). s \$ i \in S\}$

<proof>

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>

lemma *has-derivative-component*[*simp*]: — following $D ?f \mapsto ?f'$ at $?a$ within $?S$

= $(\forall i \in \text{Basis}. D (\lambda x. ?f x \cdot i) \mapsto (\lambda x. ?f' x \cdot i) \text{ at } ?a \text{ within } ?S)$
 $(D f \mapsto f' \text{ at } x \text{ within } S) \longleftrightarrow (\forall i. D (\lambda s. f s \$ i) \mapsto (\lambda s. f' s \$ i) \text{ at } x \text{ within } S)$
 $\langle \text{proof} \rangle$

lemma *has-vderiv-on-component[simp]*:
fixes $x::\text{real} \Rightarrow ('a::\text{banach}) \wedge ('n::\text{finite})$
shows $(D x = x' \text{ on } T) = (\forall i. D (\lambda t. x t \$ i) = (\lambda t. x' t \$ i) \text{ on } T)$
 $\langle \text{proof} \rangle$

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 $x_0 \equiv \text{netlimit} (\text{at } x \text{ within } T)$ **and** $m \equiv \text{real CARD}('m)$
assumes $\forall i. ((\lambda y. (f y \$ i - f x_0 \$ i - (y - x_0) *_R f' x \$ i) /_R (\|y - x_0\|)) \longrightarrow 0) (\text{at } x \text{ within } T)$
shows $((\lambda y. (f y - f x_0 - (y - x_0) *_R f' x) /_R (\|y - x_0\|)) \longrightarrow 0) (\text{at } x \text{ within } T)$
 $\langle \text{proof} \rangle$

lemma *tendsto-norm-bound*:
 $\forall x. \|G x - L\| \leq \|F x - L\| \implies (F \longrightarrow L) \text{ net} \implies (G \longrightarrow L) \text{ net}$
 $\langle \text{proof} \rangle$

lemma *tendsto-zero-norm-bound*:
 $\forall x. \|G x\| \leq \|F x\| \implies (F \longrightarrow 0) \text{ net} \implies (G \longrightarrow 0) \text{ net}$
 $\langle \text{proof} \rangle$

lemma *frechet-tendsto-vec-nth*:
fixes $f::\text{real} \Rightarrow ('a::\text{real-normed-vector}) \wedge m$
assumes $((\lambda x. (f x - f x_0 - (x - x_0) *_R f' t) /_R (\|x - x_0\|)) \longrightarrow 0) (\text{at } t \text{ within } T)$
shows $((\lambda x. (f x \$ i - f x_0 \$ i - (x - x_0) *_R f' t \$ i) /_R (\|x - x_0\|)) \longrightarrow 0) (\text{at } t \text{ within } T)$
 $\langle \text{proof} \rangle$

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* (\mathcal{P})

lemma *image-le-pred*[*simp*]: $(\mathcal{P} f A \subseteq \{s. G s\}) = (\forall x \in A. G (f x))$
 $\langle proof \rangle$

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

lemma *ivp-solsI*:
assumes $D\ X = (\lambda t. f\ t\ (X\ t))\ on\ U\ s$ **and** $X\ t_0 = s$
and $X \in U\ s \rightarrow S$ **and** $t_0 \in U\ s$
shows $X \in Sols\ f\ U\ S\ t_0\ s$
 $\langle proof \rangle$

lemma *ivp-solsD*:
assumes $X \in Sols\ f\ U\ S\ t_0\ s$
shows $D\ X = (\lambda t. f\ t\ (X\ t))\ on\ U\ s$ **and** $X\ t_0 = s$
and $X \in U\ s \rightarrow S$ **and** $t_0 \in U\ s$
 $\langle proof \rangle$

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

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

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

lemma *g-orbit-eq*:
fixes $X::('a::preorder) \Rightarrow 'b$
shows $\gamma\ X\ G\ U = \{X\ t \mid t. t \in U \wedge (\forall \tau \in down\ U\ t. G\ (X\ \tau))\}$
 $\langle proof \rangle$

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

lemma *g-orbital-eq*: $g-orbital\ f\ G\ U\ S\ t_0\ s =$

$\{X t \mid t X. t \in U s \wedge \mathcal{P} X (\text{down } (U s) t) \subseteq \{s. G s\} \wedge X \in \text{Sols } f U S t_0 s \}$
 $\langle \text{proof} \rangle$

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 $X t \in \text{g-orbital } f G U S t_0 s$
 $\langle \text{proof} \rangle$

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** $(\mathcal{P} X (\text{down } (U s) t) \subseteq \{s. G s\})$
 $\langle \text{proof} \rangle$

lemma *g-orbital f G U S t_0 s = {X t | t X. X t ∈ γ X G (U s) ∧ X ∈ Sols f U S t_0 s}*
 $\langle \text{proof} \rangle$

lemma $X \in \text{Sols } f U S t_0 s \implies \gamma X G (U s) \subseteq \text{g-orbital } f G U S t_0 s$
 $\langle \text{proof} \rangle$

lemma $\text{g-orbital } f G U S t_0 s \subseteq \text{g-orbital } f (\lambda s. \text{True}) U S t_0 s$
 $\langle \text{proof} \rangle$

no-notation *g-orbit* (γ)

3.2 Differential Invariants

definition *diff-invariant* :: $('a \Rightarrow \text{bool}) \Rightarrow (\text{real} \Rightarrow ('a :: \text{real-normed-vector}) \Rightarrow 'a) \Rightarrow$
 \Rightarrow
 $('a \Rightarrow \text{real set}) \Rightarrow 'a \text{ set} \Rightarrow \text{real} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$
where *diff-invariant I f U S t_0 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 t_0 G =*
 $(\forall s. I s \longrightarrow (\forall X \in \text{Sols } f U S t_0 s. (\forall t \in U s. (\forall \tau \in (\text{down } (U s) t). G (X \tau)) \longrightarrow I (X t))))$
 $\langle \text{proof} \rangle$

lemma *diff-inv-eq-inv-set*:

diff-invariant I f U S t_0 G = $(\forall s. I s \longrightarrow (\text{g-orbital } f G U S t_0 s) \subseteq \{s. I s\})$
 $\langle \text{proof} \rangle$

lemma *diff-invariant I f U S t_0 (λs. True) ⟹ diff-invariant I f U S t_0 G*
 $\langle \text{proof} \rangle$

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* $(\lambda s. \mu s = \nu s) f U S t_0 G$
<proof>

lemma *diff-invariant-leq-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*: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies (\forall \tau \in U(X t_0). \tau > t_0 \longrightarrow G (X \tau) \longrightarrow \mu' (X \tau) \geq \nu' (X \tau))$
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* $(\lambda s. \nu s \leq \mu s) f U S t_0 G$
<proof>

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

lemma *diff-invariant-nleq-rule*:

fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
shows *diff-invariant* $(\lambda s. \neg \nu s \leq \mu s) f U S t_0 G \iff \text{diff-invariant } (\lambda s. \nu s > \mu s) f U S t_0 G$
<proof>

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>

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

fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
assumes *Uhyp*: $\bigwedge s. s \in S \implies \text{is-interval } (U s) \bigwedge s t. s \in S \implies t \in U s \implies t_0 \leq t$
and *conts*: $\bigwedge X. (D X = (\lambda\tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies \text{continuous-on } (\mathcal{P} X$

$(U (X t_0)) \nu$
 $\wedge X. (D X = (\lambda \tau. f \tau (X \tau)) \text{ on } U(X t_0)) \implies \text{continuous-on } (\mathcal{P} X (U (X t_0))) \mu$
and $dI:\text{diff-invariant } (\lambda s. \nu s \neq \mu s) f U S t_0 G$
shows $\text{diff-invariant } (\lambda s. \nu s < \mu s) f U S t_0 G$
 $\langle \text{proof} \rangle$

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 $\text{diff-invariant } (\lambda s. I_1 s \wedge I_2 s) f U S t_0 G$
 $\langle \text{proof} \rangle$

lemma *diff-invariant-disj-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 $\text{diff-invariant } (\lambda s. I_1 s \vee I_2 s) f U S t_0 G$
 $\langle \text{proof} \rangle$

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 $f::\text{real} \implies ('a::\{\text{heine-borel}, \text{banach}\}) \implies 'a$ **and** $T::\text{real set}$ **and** $S::'a \text{ set}$ **and** $t_0::\text{real}$
assumes *open-domain*: $\text{open } T \text{ open } S$
and *interval-time*: $\text{is-interval } T$
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*: $\text{local-lipschitz } T S f$
begin

sublocale *ll-on-open-it* $T f S t_0$
 $\langle \text{proof} \rangle$

lemma *ll-on-open*: $\text{ll-on-open } T f S$
 $\langle \text{proof} \rangle$

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 $D \text{ flow } t_0 s = (\lambda t. f t (\text{flow } t_0 s t)) \text{ on } \text{ex-ivl } s$
 $\langle \text{proof} \rangle$

lemma *flow-funcset-ex-ivl*:
assumes $s \in S$
shows $\text{flow } t_0 \ s \in \text{ex-ivl } s \rightarrow S$
 $\langle \text{proof} \rangle$

lemma *flow-in-ivp-sols-ex-ivl*:
assumes $s \in S$
shows $\text{flow } t_0 \ s \in \text{Sols } f \ (\lambda s. \text{ex-ivl } s) \ S \ t_0 \ s$
 $\langle \text{proof} \rangle$

lemma *csols-eq*: $\text{csols } t_0 \ s = \{(x, t). t \in T \wedge x \in \text{Sols } f \ (\lambda s. \{t_0--t\}) \ S \ t_0 \ s\}$
 $\langle \text{proof} \rangle$

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
 $\langle \text{proof} \rangle$

lemma *unique-solution*: — proved for a subset of T for general applications
assumes $s \in S$ **and** $t_0 \in U$ **and** $t \in U$
and *is-interval* U **and** $U \subseteq \text{ex-ivl } s$
and *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 *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 $Y_1 \ t = Y_2 \ t$
 $\langle \text{proof} \rangle$

Applications of lemma *unique-solution*:

lemma *unique-solution-closed-ivl*:
assumes *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 *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 $X \ t = Y \ t$
 $\langle \text{proof} \rangle$

lemma *solution-eq-flow*:
assumes *xivp*: $D \ X = (\lambda t. f \ t \ (X \ t)) \ \text{on } \text{ex-ivl } s \ X \ t_0 = s \ X \in \text{ex-ivl } s \rightarrow S$
and $t \in \text{ex-ivl } s$ **and** $s \in S$
shows $X \ t = \text{flow } t_0 \ s \ t$
 $\langle \text{proof} \rangle$

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 $Y_1 \ t = Y_2 \ t$
 $\langle \text{proof} \rangle$

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* $f\ G\ U\ S\ t_0\ s = \text{g-orbit } Y\ G\ (U\ s)$
 $\langle \text{proof} \rangle$

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* $T\ S\ (\lambda t\ s. f1\ t\ s + f2\ t\ s)$
 $\langle \text{proof} \rangle$

lemma *picard-lindeloeff-add*: *picard-lindeloeff* $f1\ T\ S\ t_0 \implies \text{picard-lindeloeff } f2\ T\ S\ t_0 \implies$
picard-lindeloeff $(\lambda t\ s. f1\ t\ s + f2\ t\ s)\ T\ S\ t_0$
 $\langle \text{proof} \rangle$

lemma *picard-lindeloeff-constant*: *picard-lindeloeff* $(\lambda t\ s. c)\ UNIV\ UNIV\ t_0$
 $\langle \text{proof} \rangle$

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 $f :: 'a :: \{\text{heine-borel}, \text{banach}\} \Rightarrow 'a$ **and** $T\ S\ L +$
fixes $\varphi :: \text{real} \Rightarrow 'a \Rightarrow 'a$
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 $(\lambda t. \varphi\ t\ s) \in \text{Sols } (\lambda t. f)\ (\lambda s. \{0--t\})\ S\ 0\ s$
 $\langle \text{proof} \rangle$

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 $X\ t = \varphi\ t\ s$
 $\langle \text{proof} \rangle$

lemma *ex-ivl-eq*:
assumes $s \in S$

shows $ex-ivl\ s = T$
<proof>

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>

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>

lemma *has-derivative-on-open3*:
assumes $s \in S$
obtains B **where** $0 \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau. \varphi\ \tau\ s) \mapsto (\lambda\tau. \tau *_{R}\ f\ (\varphi\ 0\ s))$ **at** 0 **within** B
<proof>

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
<proof>

lemma *in-domain*:
assumes $s \in S$
shows $(\lambda t. \varphi\ t\ s) \in T \rightarrow S$
<proof>

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>

lemma *in-ivp-sols*:
assumes $s \in S$ **and** $0 \in U\ s$ **and** $U\ s \subseteq T$
shows $(\lambda t. \varphi\ t\ s) \in Sols\ (\lambda t. f)\ U\ S\ 0\ s$
<proof>

lemma *eq-solution*:
assumes $s \in S$ **and** *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $t \in U\ s$
and *xivp*: $X \in Sols\ (\lambda t. f)\ U\ S\ 0\ s$
shows $X\ t = \varphi\ t\ s$
<proof>

lemma *ivp-sols-collapse*:

assumes $T = UNIV$ **and** $s \in S$
shows $Sols (\lambda t. f) (\lambda s. T) S \ 0 \ s = \{(\lambda t. \varphi \ t \ s)\}$
 $\langle proof \rangle$

lemma *additive-in-ivp-sols*:

assumes $s \in S$ **and** $\mathcal{P} (\lambda \tau. \tau + t) T \subseteq T$
shows $(\lambda \tau. \varphi (\tau + t) s) \in Sols (\lambda t. f) (\lambda s. T) S \ 0 (\varphi (0 + t) s)$
 $\langle proof \rangle$

lemma *is-monoid-action*:

assumes $s \in S$ **and** $T = UNIV$
shows $\varphi \ 0 \ s = s$ **and** $\varphi (t_1 + t_2) s = \varphi \ t_1 (\varphi \ t_2 \ s)$
 $\langle proof \rangle$

lemma *g-orbital-collapses*:

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

definition *orbit* :: 'a \Rightarrow 'a set (γ^φ)

where $\gamma^\varphi \ s = \text{g-orbital} (\lambda t. f) (\lambda s. True) (\lambda s. T) S \ 0 \ s$

lemma *orbit-eq*:

assumes $s \in S$
shows $\gamma^\varphi \ s = \{\varphi \ t \ s \mid t. t \in T\}$
 $\langle proof \rangle$

lemma *true-g-orbit-eq*:

assumes $s \in S$
shows *g-orbit* $(\lambda t. \varphi \ t \ s) (\lambda s. True) T = \gamma^\varphi \ s$
 $\langle proof \rangle$

end

lemma *line-is-local-flow*:

$0 \in T \Longrightarrow \text{is-interval} \ T \Longrightarrow \text{open} \ T \Longrightarrow \text{local-flow} (\lambda \ s. \ c) \ T \ UNIV (\lambda \ t \ s. \ s + t \ *_R \ c)$
 $\langle proof \rangle$

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 (μ)*
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: $P \leq Q \Longrightarrow |F| P \leq |F| Q$*
 \langle *proof* \rangle

lemma *fbox-anti: $\forall s. F_1 s \subseteq F_2 s \Longrightarrow |F_2| P \leq |F_1| P$*
 \langle *proof* \rangle

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 $(\lambda s. I s \vee J s) \leq |F| (\lambda s. I s \vee J s)$
 \langle *proof* \rangle

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

lemma *fbox-eta[simp]: fbox skip P = P*
 \langle *proof* \rangle

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

lemma *fbox-test[simp]: $(\lambda s. (|\dot{i}P?| Q) s) = (\lambda s. P s \longrightarrow Q s)$*
 \langle *proof* \rangle

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

lemma *vec-upd-eq: vec-upd s i a = ($\chi\ j. if\ j = i\ then\ a\ else\ s\j)*
 \langle *proof* \rangle

definition *assign :: 'n \Rightarrow ('a \wedge n \Rightarrow 'a) \Rightarrow 'a \wedge n \Rightarrow ('a \wedge n) set ((2- ::= -) [70, 65] 61)*

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

lemma *fbox-assign[simp]*: $|x ::= e| \ Q = (\lambda s. \ Q \ (\chi \ j. \ (((\$) \ s)(x ::= (e \ s))) \ j))$
 $\langle proof \rangle$

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

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

lemma *fbox-choice*: $|(\lambda s. \ F \ s \cup \ G \ s)| \ P = (\lambda s. \ (|F| \ P) \ s \wedge \ (|G| \ P) \ s)$
 $\langle proof \rangle$

lemma *le-fbox-choice-iff*: $P \leq |(\lambda s. \ F \ s \cup \ G \ s)| \ Q \iff P \leq |F| \ Q \wedge P \leq |G| \ Q$
 $\langle proof \rangle$

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

lemma *kcomp-eq*: $(f ; g) \ x = \bigcup \ \{g \ y \ | \ y. \ y \in f \ x\}$
 $\langle proof \rangle$

lemma *fbox-kcomp[simp]*: $|G ; F| \ P = |G| \ |F| \ P$
 $\langle proof \rangle$

lemma *hoare-kcomp*:
assumes $P \leq |G| \ R \ R \leq |F| \ Q$
shows $P \leq |G ; F| \ Q$
 $\langle proof \rangle$

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

lemma *fbox-if-then-else[simp]*:
 $|IF \ T \ THEN \ X \ ELSE \ Y| \ Q = (\lambda s. \ (T \ s \longrightarrow \ (|X| \ Q) \ s) \wedge \ (\neg \ T \ s \longrightarrow \ (|Y| \ Q) \ s))$
 $\langle proof \rangle$

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 $P \leq |IF \ T \ THEN \ X \ ELSE \ Y| \ Q$
 $\langle proof \rangle$

definition *kpower* :: $('a \Rightarrow 'a \ set) \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow 'a \ set)$

where $kpower\ f\ n = (\lambda s. ((;) f \sim n)\ skip\ s)$

lemma *kpower-base*:

shows $kpower\ f\ 0\ s = \{s\}$ **and** $kpower\ f\ (Suc\ 0)\ s = f\ s$
 $\langle proof \rangle$

lemma *kpower-simp*: $kpower\ f\ (Suc\ n)\ s = (f ; kpower\ f\ n)\ s$

$\langle proof \rangle$

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 $\forall s. I\ s \longrightarrow (\forall s'. s' \in (kpower\ F\ n\ s) \longrightarrow I\ s')$

$\langle proof \rangle$

lemma *kstar-inv*: $I \leq |F| I \Longrightarrow I \leq |F^*| I$

$\langle proof \rangle$

lemma *fbx-kstarI*:

assumes $P \leq I$ **and** $I \leq Q$ **and** $I \leq |F| I$

shows $P \leq |F^*| Q$

$\langle proof \rangle$

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*: $LOOP\ X\ INV\ G = LOOP\ X\ INV\ I$

$\langle proof \rangle$

lemma *fbx-loopI*: $P \leq I \Longrightarrow I \leq Q \Longrightarrow I \leq |F| I \Longrightarrow P \leq |LOOP\ F\ INV\ I| Q$

$\langle proof \rangle$

lemma *wp-loopI-break*:

$P \leq |Y| I \Longrightarrow I \leq |X| I \Longrightarrow I \leq Q \Longrightarrow P \leq |Y ; (LOOP\ X\ INV\ I)| Q$

$\langle proof \rangle$

4.2 Verification of hybrid programs

Verification by providing evolution

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

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

lemma *fbx-g-evol[simp]*:

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

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

Verification by providing solutions

lemma *fbox-g-orbital*: $|x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] Q =$
 $(\lambda s. \forall X \in \text{Sols } f \ U \ S \ t_0 \ s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))$
 $\langle \text{proof} \rangle$

context *local-flow*
begin

lemma *fbox-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 $|x'=(\lambda t. f) \ \& \ G \ \text{on} \ U \ S \ @ \ 0] Q =$
 $(\lambda s. s \in S \longrightarrow (\forall t \in (U s). (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle \text{proof} \rangle$

lemma *fbox-g-ode*: $|x'=(\lambda t. f) \ \& \ G \ \text{on} \ (\lambda s. T) \ S \ @ \ 0] Q =$
 $(\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle \text{proof} \rangle$

lemma *fbox-g-ode-ivl*: $t \geq 0 \implies t \in T \implies |x'=(\lambda t. f) \ \& \ G \ \text{on} \ (\lambda s. \{0..t\}) \ S \ @$
 $0] Q =$
 $(\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle \text{proof} \rangle$

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

end

Verification with differential invariants

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

lemma *fbox-g-orbital-guard*:

assumes $H = (\lambda s. G s \wedge Q s)$
shows $|x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] Q = |x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] H$
 $\langle \text{proof} \rangle$

lemma *fbox-g-orbital-inv*:

assumes $P \leq I$ **and** $I \leq |x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] I$ **and** $I \leq Q$
shows $P \leq |x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] Q$
 $\langle \text{proof} \rangle$

lemma *fbox-diff-inv[simp]*:

$(I \leq |x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] \ I) = \text{diff-invariant } I \ f \ U \ S \ t_0 \ G$
 $\langle \text{proof} \rangle$

lemma *diff-inv-guard-ignore*:

assumes $I \leq |x'=f \ \& \ (\lambda s. \ \text{True}) \ \text{on} \ U \ S \ @ \ t_0] \ I$
shows $I \leq |x'=f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0] \ I$
 $\langle \text{proof} \rangle$

context *local-flow*

begin

lemma *fbx-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 $\text{diff-invariant } I \ (\lambda t. \ f) \ U \ S \ 0 \ (\lambda s. \ \text{True}) =$
 $((\lambda s. s \in S \implies I \ s) = |x'= (\lambda t. \ f) \ \& \ (\lambda s. \ \text{True}) \ \text{on} \ U \ S \ @ \ 0] \ (\lambda s. s \in S \implies I$
 $s))$
 $\langle \text{proof} \rangle$

lemma *diff-inv-eq-inv-set*:

$\text{diff-invariant } I \ (\lambda t. \ f) \ (\lambda s. \ T) \ S \ 0 \ (\lambda s. \ \text{True}) = (\forall s. I \ s \implies \gamma^\varphi \ s \subseteq \{s. I \ s\})$
 $\langle \text{proof} \rangle$

end

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

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

abbreviation *g-dl-ode-inv* :: $((\text{'a}::\text{banach}) \Rightarrow \text{'a}) \Rightarrow \text{'a} \ \text{pred} \Rightarrow \text{'a} \ \text{pred} \Rightarrow \text{'a} \Rightarrow \text{'a} \ \text{set}$
 $((\text{!}x' = - \ \& \ - \ \text{DINV } -))$
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 $|x'=f \ \& \ G] \ Q =$
 $(\lambda s. \ \forall t \geq 0. \ (\forall \tau \in \{0..t\}. \ G \ (\varphi \ \tau \ s)) \implies Q \ (\varphi \ t \ s))$
 $\langle \text{proof} \rangle$

lemma *diff-solve-axiom2*:

fixes $c::'a::\{heine-borel, banach\}$
shows $|x'=(\lambda s. c) \ \& \ G| \ Q =$
 $(\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (s + \tau *_{R} c))) \longrightarrow Q (s + t *_{R} c)$
 $\langle proof \rangle$

lemma *diff-solve-rule:*

assumes *local-flow* $f \ UNIV \ UNIV \ \varphi$
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 $P \leq |x' = f \ \& \ G| \ Q$
 $\langle proof \rangle$

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

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

lemma *diff-weak-rule:* $G \leq Q \implies P \leq |x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0| \ Q$
 $\langle proof \rangle$

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

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

lemma *diff-cut-axiom:*

assumes $|x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0| \ C = (\lambda s. True)$
shows $|x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0| \ Q = |x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \ \text{on } U \ S \ @ \ t_0| \ Q$
 $\langle proof \rangle$

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 $P \leq |x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0| \ Q$
 $\langle proof \rangle$

lemma *diff-inv-axiom1:*

assumes $G \ s \longrightarrow I \ s$ **and** *diff-invariant* $I (\lambda t. f) (\lambda s. \{t. t \geq 0\}) \ UNIV \ 0 \ G$
shows $(|x' = f \ \& \ G| \ I) \ s$
 $\langle proof \rangle$

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 $|x' = f \ \& \ G| \ I = |(\lambda s. \{x. s = x \wedge G \ s\})| \ I$
 $\langle proof \rangle$

lemma *diff-inv-rule*:
assumes $P \leq I$ **and** *diff-invariant* $I f U S t_0 G$ **and** $I \leq Q$
shows $P \leq [x' = f \ \& \ G \text{ on } U S @ t_0] Q$
<proof>

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 $x' t = y t$ 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* :: $real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
where $\varphi t s \equiv (\chi \ i. \text{if } i = 1 \text{ then } s\$1 * \cos t + s\$2 * \sin t \text{ else } -s\$1 * \sin t + s\$2 * \cos t)$

— Verified with annotated dynamics.

lemma *pendulum-dyn*: $(\lambda s. r^2 = (s\$1)^2 + (s\$2)^2) \leq [EVOL \ \varphi \ G \ T] (\lambda s. r^2 = (s\$1)^2 + (s\$2)^2)$
<proof>

lemma *pendulum-inv*: $(\lambda s. r^2 = (s\$1)^2 + (s\$2)^2) \leq [x' = f \ \& \ G] (\lambda s. r^2 = (s\$1)^2 + (s\$2)^2)$
<proof>

lemma *local-flow-pend*: *local-flow* $f \ UNIV \ UNIV \ \varphi$
<proof>

lemma *pendulum-flow*: $(\lambda s. r^2 = (s\$1)^2 + (s\$2)^2) \leq [x' = f \ \& \ G] (\lambda s. r^2 = (s\$1)^2 + (s\$2)^2)$
<proof>

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* :: $real \Rightarrow real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
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>

lemma *diff-invariant* $(\lambda s. 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)$ $(\lambda t. f g)$
 $(\lambda s. UNIV) S t_0 G$
<proof>

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
 $= 0)) ;$
 $(IF (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)]$
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$
<proof>

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>

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⟩

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 $(\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)$
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$
 ⟨proof⟩

lemma *local-flow-ball*: *local-flow* ($f g$) UNIV UNIV (φg)
 ⟨proof⟩

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

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

abbreviation *temp-vec-field* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$ (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 φ 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*: $0 < a \implies \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1 - s_2|$
 $\langle proof \rangle$

lemma *local-lipschitz-temp-dyn*:
assumes $0 < (a::real)$
shows *local-lipschitz UNIV UNIV* $(\lambda t::real. f\ a\ L)$
 $\langle proof \rangle$

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

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** $\exp(-a * t) * T \leq T_{max}$
 $\langle proof \rangle$

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 $T_{min} \leq L - \exp(-(a * t)) * (L - T)$
 $\langle proof \rangle$

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

lemma *thermostat*:
assumes $a > 0$ **and** $0 < T_{min}$ **and** $T_{max} < L$
shows $(\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max} \wedge s\$4 = 0) \leq$
 $|LOOP$
— control
 $((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq T_{min} + 1) THEN (4 ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq T_{max} - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip));$
— dynamics
 $(IF (\lambda s. s\$4 = 0) THEN (x' = f\ a\ 0 \ \&\ (\lambda s. s\$2 \leq -(\ln(T_{min}/s\$3))/a))$
 $ELSE (x' = f\ a\ L \ \&\ (\lambda s. s\$2 \leq -(\ln((L - T_{max})/(L - s\$3))/a))))$
 $INV (\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max} \wedge (s\$4 = 0 \vee s\$4 = 1))$
 $(\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max})$
 $\langle proof \rangle$

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* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (\varphi)$
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 $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k * s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

lemma *local-flow-tank*: *local-flow* ($f\ k$) UNIV UNIV ($\varphi\ k$)
<proof>

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 $y \leq hmax \implies y - c_o * \tau \leq hmax$
<proof>

lemma *tank-flow*:

assumes $0 < c_o$ **and** $c_o < c_i$
shows $I\ hmin\ hmax \leq$
 $|LOOP$
— 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 (λs. s$4 = 0) THEN (x' = f (ci - co) & (G hmax (ci - co)))
ELSE (x' = f (-co) & (G hmin (-co)))) INV I hmin hmax
I hmin hmax
⟨proof⟩

```

```

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 → 60)
  and dagger († [101] 100)
  and Relation.relcomp (infixl ; 75)
  and eta (η)
  and kcomp (infixl ∘K 75)

```

```

type-synonym 'a pred = 'a ⇒ bool

```

```

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

```

5.1 Verification of regular programs

Properties of the forward box operator.

```

lemma fbℱ F S = (∩ ∘ ℙ (- opK F)) (- S)
  ⟨proof⟩

```

```

lemma fbℱ F S = {s. F s ⊆ S}
  ⟨proof⟩

```

lemma *ffb-eq*: $fb_{\mathcal{F}} F S = \{s. \forall s'. s' \in F s \longrightarrow s' \in S\}$
 ⟨proof⟩

lemma *ffb-iso*: $P \leq Q \implies fb_{\mathcal{F}} F P \leq fb_{\mathcal{F}} F Q$
 ⟨proof⟩

lemma *ffb-invariants*:

assumes $\{s. I s\} \leq fb_{\mathcal{F}} F \{s. I s\}$ **and** $\{s. J s\} \leq fb_{\mathcal{F}} F \{s. J s\}$
shows $\{s. I s \wedge J s\} \leq fb_{\mathcal{F}} F \{s. I s \wedge J s\}$
and $\{s. I s \vee J s\} \leq fb_{\mathcal{F}} F \{s. I s \vee J s\}$
 ⟨proof⟩

lemma *ffb-skip[simp]*: $fb_{\mathcal{F}} skip S = S$
 ⟨proof⟩

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

lemma *ffb-test[simp]*: $fb_{\mathcal{F}} \dot{\iota} P? Q = \{s. P s \longrightarrow s \in Q\}$
 ⟨proof⟩

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

lemma *ffb-assign[simp]*: $fb_{\mathcal{F}} (x ::= e) Q = \{s. (\chi \ j. (((\$) s)(x := (e \ s))) \ j) \in Q\}$
 ⟨proof⟩

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

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

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

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

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

lemma *hoare-kcomp*:

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

<proof>

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

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

$\text{fb}_{\mathcal{F}} (IF \ T \ THEN \ X \ ELSE \ Y) \ Q = \{s. T \ s \longrightarrow s \in \text{fb}_{\mathcal{F}} \ X \ Q\} \cap \{s. \neg T \ s \longrightarrow s \in \text{fb}_{\mathcal{F}} \ Y \ Q\}$
<proof>

lemma *hoare-if-then-else*:

assumes $P \cap \{s. T \ s\} \leq \text{fb}_{\mathcal{F}} \ X \ Q$
and $P \cap \{s. \neg T \ s\} \leq \text{fb}_{\mathcal{F}} \ Y \ Q$
shows $P \leq \text{fb}_{\mathcal{F}} (IF \ T \ THEN \ X \ ELSE \ Y) \ Q$
<proof>

lemma *kpower-inv*: $I \leq \{s. \forall y. y \in F \ s \longrightarrow y \in I\} \Longrightarrow I \leq \{s. \forall y. y \in (\text{kpower } F \ n \ s) \longrightarrow y \in I\}$
<proof>

lemma *kstar-inv*: $I \leq \text{fb}_{\mathcal{F}} \ F \ I \Longrightarrow I \subseteq \text{fb}_{\mathcal{F}} (\text{kstar } F) \ I$
<proof>

lemma *ffb-kstarI*:

assumes $P \leq I$ **and** $I \leq Q$ **and** $I \leq \text{fb}_{\mathcal{F}} \ F \ I$
shows $P \leq \text{fb}_{\mathcal{F}} (\text{kstar } F) \ Q$
<proof>

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
<proof>

lemma *ffb-loopI*: $P \leq \{s. I \ s\} \Longrightarrow \{s. I \ s\} \leq Q \Longrightarrow \{s. I \ s\} \leq \text{fb}_{\mathcal{F}} \ F \ \{s. I \ s\} \Longrightarrow P \leq \text{fb}_{\mathcal{F}} (LOOP \ F \ INV \ I) \ Q$
<proof>

lemma *ffb-loopI-break*:

$P \leq \text{fb}_{\mathcal{F}} \ Y \ \{s. I \ s\} \Longrightarrow \{s. I \ s\} \leq \text{fb}_{\mathcal{F}} \ X \ \{s. I \ s\} \Longrightarrow \{s. I \ s\} \leq Q \Longrightarrow P \leq \text{fb}_{\mathcal{F}} (Y ; (LOOP \ X \ INV \ I)) \ Q$
<proof>

5.2 Verification of hybrid programs

Verification by providing evolution

definition *g-evol* :: (('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b pred \Rightarrow ('b \Rightarrow 'a set) \Rightarrow ('b \Rightarrow 'b

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

lemma *fbx-g-evol[simp]*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $fb_{\mathcal{F}} (EVOL \varphi G U) Q = \{s. (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow (\varphi t s) \in Q)\}$
 $\langle \text{proof} \rangle$

Verification by providing solutions

lemma *ffb-g-orbital*: $fb_{\mathcal{F}} (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) Q =$
 $\{s. \forall X \in \text{Sols } f \ U \ S \ t_0 \ s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow (X t) \in Q\}$
 $\langle \text{proof} \rangle$

context *local-flow*
begin

lemma *ffb-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 $fb_{\mathcal{F}} (x' = (\lambda t. f) \ \& \ G \ \text{on } U \ S \ @ \ 0) Q =$
 $\{s. s \in S \longrightarrow (\forall t \in (U s). (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow (\varphi t s) \in Q)\}$
 $\langle \text{proof} \rangle$

lemma *ffb-g-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)
 $\langle \text{proof} \rangle$

lemma *ffb-g-ode-ivl*: $t \geq 0 \implies t \in T \implies fb_{\mathcal{F}} (x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. \{0..t\}) \ S$
 $@ \ 0) Q =$
 $\{s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow (\varphi t s) \in Q)\}$
 $\langle \text{proof} \rangle$

lemma *ffb-orbit*: $fb_{\mathcal{F}} \gamma^{\varphi} Q = \{s. s \in S \longrightarrow (\forall t \in T. \varphi t s \in Q)\}$
 $\langle \text{proof} \rangle$

end

Verification with differential invariants

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

lemma *ffb-g-orbital-guard*:
assumes $H = (\lambda s. G s \wedge Q s)$
shows $fb_{\mathcal{F}} (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \{s. Q s\} = fb_{\mathcal{F}} (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0)$
 $\{s. H s\}$
 $\langle \text{proof} \rangle$

lemma *ffb-g-orbital-inv*:

assumes $P \leq I$ **and** $I \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ I$ **and** $I \leq Q$

shows $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ Q$

<proof>

lemma *ffb-diff-inv[simp]*:

$(\{s. I \ s\} \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{s. I \ s\}) = \text{diff-invariant } I \ f \ U \ S \ t_0 \ G$

<proof>

lemma *bdf-diff-inv*:

$\text{diff-invariant } I \ f \ U \ S \ t_0 \ G = (\text{bd}_{\mathcal{F}} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{s. I \ s\} \leq \{s. I \ s\})$

<proof>

lemma *diff-inv-guard-ignore*:

assumes $\{s. I \ s\} \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \text{True}) \ \text{on} \ U \ S \ @ \ t_0) \ \{s. I \ s\}$

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

<proof>

context *local-flow*

begin

lemma *ffb-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 $\text{diff-invariant } I \ (\lambda t. f) \ U \ S \ 0 \ (\lambda s. \text{True}) =$

$(\{s. s \in S \implies I \ s\} = \text{fb}_{\mathcal{F}} (x' = (\lambda t. f) \ \& \ (\lambda s. \text{True}) \ \text{on} \ U \ S \ @ \ 0) \ \{s. s \in S \implies I \ s\})$

<proof>

lemma *diff-inv-eq-inv-set*:

$\text{diff-invariant } I \ (\lambda t. f) \ (\lambda s. T) \ S \ 0 \ (\lambda s. \text{True}) = (\forall s. I \ s \implies \gamma^{\varphi} \ s \subseteq \{s. I \ s\})$

<proof>

end

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

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

<proof>

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

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

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

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

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

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

$\langle \text{proof} \rangle$

lemma diff-solve-axiom2 :

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

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

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

$\langle \text{proof} \rangle$

lemma diff-solve-rule :

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

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

$\langle \text{proof} \rangle$

lemma diff-weak-axiom1 : $s \in (\text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } U S @ t_0) \{s. G s\})$

$\langle \text{proof} \rangle$

lemma diff-weak-axiom2 : $\text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } T S @ t_0) Q = \text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } T S @ t_0) \{s. G s \longrightarrow s \in Q\}$

$\langle \text{proof} \rangle$

lemma diff-weak-rule : $\{s. G s\} \leq Q \implies P \leq \text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } T S @ t_0) Q$

$\langle \text{proof} \rangle$

lemma $\text{ffb-g-orbital-eq-univD}$:

assumes $\text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } U S @ t_0) \{s. C s\} = \text{UNIV}$

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

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

$\langle \text{proof} \rangle$

lemma diff-cut-axiom :

assumes $\text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } U S @ t_0) \{s. C s\} = \text{UNIV}$

shows $\text{fb}_{\mathcal{F}} (x' = f \& G \text{ on } U S @ t_0) Q = \text{fb}_{\mathcal{F}} (x' = f \& (\lambda s. G s \wedge C s) \text{ on } U S @ t_0) Q$

$\langle \text{proof} \rangle$

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>

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 $s \in (fb_{\mathcal{F}}\ (x' = f \ \&\ G)\ \{s. I\ s\})$

<proof>

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 $fb_{\mathcal{F}}\ (x' = f \ \&\ G)\ \{s. I\ s\} = fb_{\mathcal{F}}\ (\lambda s. \{x. s = x \wedge G\ s\})\ \{s. I\ s\}$

<proof>

lemma *diff-inv-rule*:

assumes $P \leq \{s. I\ s\}$ **and** *diff-invariant* $I\ f\ U\ S\ t_0\ G$ **and** $\{s. I\ s\} \leq Q$
shows $P \leq fb_{\mathcal{F}}\ (x' = f \ \&\ G\ \text{on}\ U\ S\ @\ t_0)\ Q$

<proof>

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 $x' = -y$ 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* :: $real^2 \Rightarrow real^2\ (f)$

where $f\ s \equiv (\chi\ i. \text{if } i = 1 \text{ then } s2 \text{ else } -s1)$

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

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 by providing the dynamics

lemma *pendulum-dyn*: $\{s. r^2 = (s1)^2 + (s2)^2\} \leq fb_{\mathcal{F}}\ (EVOL\ \varphi\ G\ T)\ \{s. r^2 = (s1)^2 + (s2)^2\}$

<proof>

lemma *pendulum-inv*: $\{s. r^2 = (s\$1)^2 + (s\$2)^2\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \{s. r^2 = (s\$1)^2 + (s\$2)^2\}$
 ⟨proof⟩

lemma *local-flow-pend*: *local-flow* f *UNIV UNIV* φ
 ⟨proof⟩

lemma *pendulum-flow*: $\{s. r^2 = (s\$1)^2 + (s\$2)^2\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \{s. r^2 = (s\$1)^2 + (s\$2)^2\}$
 ⟨proof⟩

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

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

— 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⟩

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

<proof>

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>

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>

lemma *bouncing-ball-dyn*: $g < 0 \implies h \geq 0 \implies$

$\{s. s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$

(*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 * g * s\$1 = 2 * g * h + s\$2 * s\2))

$\{s. 0 \leq s\$1 \wedge s\$1 \leq h\}$

<proof>

lemma *local-flow-ball*: *local-flow* (*f g*) *UNIV UNIV* (φ *g*)

<proof>

lemma *bouncing-ball-flow*: $g < 0 \implies h \geq 0 \implies$

$\{s. s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$

(*LOOP* (

($x' = (\lambda t. f g)$ & ($\lambda s. s\$1 \geq 0$) *on* ($\lambda s. UNIV$) *UNIV @ 0*) ;

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

INV ($\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\2))

$\{s. 0 \leq s\$1 \wedge s\$1 \leq h\}$

<proof>

no-notation *fball* (*f*)

and *ball-flow* (φ)

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 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* :: $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (\varphi)$

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>

lemma *local-lipschitz-temp-dyn*:

assumes $0 < (a::real)$

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

<proof>

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

<proof>

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>

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

assumes $a > 0$ **and** *Thyps*: $Tmin \leq T \ T \leq Tmax \ Tmax < (L::real)$

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

shows $L - Tmax \leq \exp(-(a * t)) * (L - T)$

and $L - \exp(-(a * t)) * (L - T) \leq Tmax$

and $Tmin \leq L - \exp(-(a * t)) * (L - T)$

<proof>

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

lemma *thermostat*:

assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$

shows $\{s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0\} \leq fb_{\mathcal{F}}$

(LOOP

— control

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

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

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

— dynamics

$(IF (\lambda s. s\$4 = 0) THEN (x' = (\lambda t. f a 0) \ \& \ (\lambda s. s\$2 \leq -(\ln (Tmin/s\$3))/a)$

$on (\lambda s. \{0..t\}) UNIV @ 0)$
 $ELSE (x'=(\lambda t. f a L) \& (\lambda s. s\$2 \leq - (\ln ((L-Tmax)/(L-s\$3)))/a) on (\lambda s.$
 $\{0..t\}) UNIV @ 0)))$
 $INV (\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1))$
 $\{s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax\}$
 $\langle proof \rangle$

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 ($s\$4 = 1$) 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)
 $\langle proof \rangle$

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

<proof>

lemma *tank-flow*:

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

shows $\text{Collect } (I \text{ hmin hmax}) \leq \text{fb}_{\mathcal{F}}$

(LOOP

— control

$((\mathcal{I} ::= (\lambda s. 0)); (\mathcal{J} ::= (\lambda s. s\$1)));$

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

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

— dynamics

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

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

$(\text{Collect } (I \ \text{hmin} \ \text{hmax}))$

<proof>

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* ($(\text{if } (-) / \text{then } (-) / \text{else } (-)) [0, 0, 10] 10$)

notation *zero-class.zero* (0)

context *antidomain-kleene-algebra*

begin

— Skip

lemma $|1]$ $x = d x$
 $\langle proof \rangle$

lemma $|0]$ $q = 1$
 $\langle proof \rangle$

lemma $|x \cdot y]$ $q = |x]$ $|y]$ q
 $\langle proof \rangle$

declare *fbox-mult* [*simp*]

— Nondeterministic choice

lemma $|x + y]$ $q = |x]$ $q \cdot |y]$ q
 $\langle proof \rangle$

lemma *le-fbox-choice-iff*: $d p \leq |x + y]q \iff (d p \leq |x]q) \wedge (d p \leq |y]q)$
 $\langle proof \rangle$

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
 $\langle proof \rangle$

lemma *fbox-cond* [*simp*]: $|if p then x else y]$ $q = (ad p + |x]$ $q) \cdot (d p + |y]$ $q)$
 $\langle proof \rangle$

lemma *fbox-cond2*: $|if p then x else y]$ $q = (d p \cdot |x]$ $q) + (ad p \cdot |y]$ $q)$ (**is** *?lhs = ?d1 + ?d2*)
 $\langle proof \rangle$

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*: $d p \cdot x \leq x \cdot d p \implies d q \leq |x]$ $r \implies d p \cdot d q \leq |x]$ $(d p \cdot d r)$
 $\langle proof \rangle$

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

lemma *fbox-export2*: $|x]$ $p \leq |x \cdot ad q]$ $(d p \cdot ad q)$
 $\langle proof \rangle$

lemma *fbox-while*: $d p \cdot d t \leq |x]$ $p \implies d p \leq |(d t \cdot x)^* \cdot ad t]$ $(d p \cdot ad t)$
 $\langle proof \rangle$

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>

lemma *fbox-seq-var*: $p \leq |x| p' \implies p' \leq |y| q \implies p \leq |x \cdot y| q$

<proof>

lemma *fbox-whilei-break*:

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

<proof>

definition *aka-loopi* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*loop - inv - [64,64] 63*)

where $loop x inv i = x^*$

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

<proof>

lemma *fbox-loopi*: $p \leq d i \implies d i \leq |x| i \implies d i \leq d q \implies p \leq |loop x inv i| q$

<proof>

lemma *fbox-loopi-break*:

$p \leq |y| d i \implies d i \leq |x| i \implies d i \leq d q \implies p \leq |y \cdot (loop x inv i)| q$

<proof>

lemma $p \leq i \implies i \leq |x|i \implies i \leq q \implies p \leq |x|q$

<proof>

lemma $p \leq d i \implies d i \leq |x|i \implies i \leq d q \implies p \leq |x|q$

<proof>

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

<proof>

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

<proof>

lemma *plus-inv*: $i \leq |x| i \implies j \leq |x| j \implies (i + j) \leq |x| (i + j)$

<proof>

lemma *mult-inv*: $d i \leq |x| i \implies d j \leq |x| j \implies (d i \cdot d j) \leq |x| (d i \cdot d j)$

<proof>

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 *diod-one-zero*

begin

lemma *power-inductl*: $z + x \cdot y \leq y \implies (x \hat{\ } n) \cdot z \leq y$
<proof>

lemma *power-inductr*: $z + y \cdot x \leq y \implies z \cdot (x \hat{\ } n) \leq y$
<proof>

end

interpretation *rel-diod*: *diod-one-zero* (\cup) (O) *Id* $\{\}$ (\subseteq) (\subset)
<proof>

lemma *power-is-relpow*: $\text{rel-diod.power } X \ n = X \hat{\ } n$
<proof>

lemma *rel-star-def*: $X \hat{\ } * = (\bigcup n. \text{rel-diod.power } X \ n)$
<proof>

lemma *rel-star-contl*: $X \ O \ Y \hat{\ } * = (\bigcup n. X \ O \ \text{rel-diod.power } Y \ n)$
<proof>

lemma *rel-star-contr*: $X \hat{\ } * \ O \ Y = (\bigcup n. (\text{rel-diod.power } X \ n) \ O \ Y)$
<proof>

interpretation *rel-ka*: *kleene-algebra* (\cup) (O) *Id* $\{\}$ (\subseteq) (\subset) *rtrancl*
<proof>

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

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

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 ($(1[\cdot])$) **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-ad $\lceil P \rceil = \lceil \lambda s. \neg P s \rceil$
rel-aka.ads-d $\lceil P \rceil = \lceil P \rceil$
 <proof>

lemma *in-p2r* [*simp*]: $(a,b) \in \lceil P \rceil = (P a \wedge a = b)$
 <proof>

Lemmas for verification condition generation

lemma *wp-rel*: $wp R \lceil P \rceil = \lceil \lambda x. \forall y. (x,y) \in R \longrightarrow P y \rceil$
 <proof>

lemma *wp-test*[*simp*]: $wp \lceil P \rceil \lceil Q \rceil = \lceil \lambda s. P s \longrightarrow Q s \rceil$
 <proof>

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

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

lemma *wp-assign [simp]*: $\text{wp } (x ::= e) \ [\![Q]\!] = [\!\lambda s. Q \ (\chi \ j. (((\$) \ s)(x ::= (e \ s))) \ j)]$
 $\langle \text{proof} \rangle$

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

lemma *wp-nondet-assign [simp]*: $\text{wp } (x ::= ?) \ [\![P]\!] = [\!\lambda s. \forall k. P \ (\chi \ j. (((\$) \ s)(x ::= k)) \ j)]$
 $\langle \text{proof} \rangle$

lemma *le-wp-choice-iff*: $[\![P]\!] \leq \text{wp } (X \cup Y) \ [\![Q]\!] \iff [\![P]\!] \leq \text{wp } X \ [\![Q]\!] \wedge [\![P]\!] \leq \text{wp } Y \ [\![Q]\!]$
 $\langle \text{proof} \rangle$

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 $IF \ P \ THEN \ X \ ELSE \ Y \equiv \text{rel-aka.aka-cond } [\![P]\!] \ X \ Y$

— Finite iteration

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

lemma *change-loopI*: $LOOP \ X \ INV \ G = LOOP \ X \ INV \ I$
 $\langle \text{proof} \rangle$

lemma *wp-loopI*:
 $[\![P]\!] \leq [\![I]\!] \implies [\![I]\!] \leq [\![Q]\!] \implies [\![I]\!] \leq \text{wp } R \ [\![I]\!] \implies [\![P]\!] \leq \text{wp } (LOOP \ R \ INV \ I) \ [\![Q]\!]$
 $\langle \text{proof} \rangle$

lemma *wp-loopI-break*:
 $[\![P]\!] \leq \text{wp } Y \ [\![I]\!] \implies [\![I]\!] \leq \text{wp } X \ [\![I]\!] \implies [\![I]\!] \leq [\![Q]\!] \implies [\![P]\!] \leq \text{wp } (Y ; (LOOP \ X \ INV \ I)) \ [\![Q]\!]$
 $\langle \text{proof} \rangle$

6.3.2 Evolution commands

Verification by providing evolution

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

lemma *wp-g-dyn [simp]*:
fixes $\varphi :: ('a :: \text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{wp } (EVOL \ \varphi \ G \ U) \ [\![Q]\!] = [\!\lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. \ G \ (\varphi \ \tau \ s))\]$

$\longrightarrow Q (\varphi t s)]$
 $\langle proof \rangle$

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] =

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

$\langle proof \rangle$

context *local-flow*

begin

lemma *wp-g-ode-subset*:

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

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

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

$\langle proof \rangle$

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

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

$\langle proof \rangle$

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

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

$\langle proof \rangle$

lemma *wp-orbit*: wp ({(s,s') | s s'. s' $\in \gamma^\varphi$ s}) [Q] = [$\lambda s. s \in S \longrightarrow (\forall t \in T. Q$

($\varphi t s$)]

$\langle proof \rangle$

end

Verification with differential invariants

definition *g-ode-inv* :: (real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a pred \Rightarrow ('a \Rightarrow real set) \Rightarrow 'a set \Rightarrow

real \Rightarrow 'a pred \Rightarrow 'a rel ((1x'=- & - on - - @ - DINV -))

where (x' = f & G on U S @ t₀ DINV I) = (x' = f & G on U S @ t₀)

lemma *wp-g-orbital-guard*:

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

shows wp (x' = f & G on U S @ t₀) [Q] = wp (x' = f & G on U S @ t₀) [H]

$\langle proof \rangle$

lemma *wp-g-orbital-inv*:

assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil I \rceil$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$
<proof>

lemma *wp-diff-inv[simp]*: $(\lceil I \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil I \rceil) = \text{diff-invariant } I \ f \ U \ S \ t_0 \ G$
<proof>

lemma *diff-inv-guard-ignore*:
assumes $\lceil I \rceil \leq wp(x' = f \ \& \ (\lambda s. \text{True}) \text{ on } U \ S \ @ \ t_0) \lceil I \rceil$
shows $\lceil I \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil I \rceil$
<proof>

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 $\text{diff-invariant } I \ (\lambda t. f) \ U \ S \ 0 \ (\lambda s. \text{True}) = (\lceil \lambda s. s \in S \longrightarrow I \ s \rceil = wp(x' = (\lambda t. f) \ \& \ (\lambda s. \text{True}) \text{ on } U \ S \ @ \ 0) \lceil \lambda s. s \in S \longrightarrow I \ s \rceil)$
<proof>

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

end

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

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* :: $((\text{'a}::\text{banach}) \Rightarrow \text{'a}) \Rightarrow \text{'a pred} \Rightarrow \text{'a rel} ((\text{!}x' = - \ \& \ -))$
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* :: $((\text{'a}::\text{banach}) \Rightarrow \text{'a}) \Rightarrow \text{'a pred} \Rightarrow \text{'a pred} \Rightarrow \text{'a rel} ((\text{!}x' = - \ \& \ - \ \text{DINV } -))$
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* *UNIV* *UNIV* φ

shows $wp (x' = f \ \& \ G) \lceil Q \rceil =$
 $\lceil \lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s) \rceil$
 $\langle proof \rangle$

lemma *diff-solve-axiom2*:

fixes $c :: 'a :: \{heine-borel, banach\}$
shows $wp (x' = (\lambda s. c) \ \& \ G) \lceil Q \rceil =$
 $\lceil \lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (s + \tau *_{\mathbb{R}} c)) \longrightarrow Q (s + t *_{\mathbb{R}} c) \rceil$
 $\langle proof \rangle$

lemma *diff-solve-rule*:

assumes *local-flow* $f \ UNIV \ UNIV \ \varphi$
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 $\lceil P \rceil \leq wp (x' = f \ \& \ G) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *diff-weak-axiom1*: $Id \subseteq wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil G \rceil$
 $\langle proof \rangle$

lemma *diff-weak-axiom2*:

$wp (x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \lceil Q \rceil = wp (x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \lceil \lambda s. G \ s \longrightarrow Q \ s \rceil$
 $\langle proof \rangle$

lemma *diff-weak-rule*:

assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *wp-g-evol-IdD*:

assumes $wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil C \rceil = Id$
and $\forall \tau \in (\text{down } (U \ s) \ t). (s, x \ \tau) \in (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0)$
shows $\forall \tau \in (\text{down } (U \ s) \ t). C (x \ \tau)$
 $\langle proof \rangle$

lemma *diff-cut-axiom*:

assumes $wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil C \rceil = Id$
shows $wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil = wp (x' = f \ \& \ (\lambda s. G \ s \ \wedge \ C \ s) \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *diff-cut-rule*:

assumes $wp\text{-}C: \lceil P \rceil \leq wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil C \rceil$
and $wp\text{-}Q: \lceil P \rceil \leq wp (x' = f \ \& \ (\lambda s. G \ s \ \wedge \ C \ s) \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil$
shows $\lceil P \rceil \leq wp (x' = f \ \& \ G \ \text{on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

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 $(s,s) \in wp (x' = f \ \& \ G) \lceil I \rceil$
 $\langle proof \rangle$

lemma *diff-inv-axiom2*:

assumes *picard-lindeloeff* $(\lambda t. f) \text{ UNIV UNIV } 0$
and $\bigwedge s. \{t::real. t \geq 0\} \subseteq \text{picard-lindeloeff.ex-ivl} (\lambda t. f) \text{ UNIV UNIV } 0 s$
and *diff-invariant* $I (\lambda t. f) (\lambda s. \{t::real. t \geq 0\}) \text{ UNIV } 0 G$
shows $wp (x' = f \ \& \ G) \lceil I \rceil = wp \lceil G \rceil \lceil I \rceil$
 $\langle proof \rangle$

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

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

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

— Verified by providing dynamics.

lemma *pendulum-dyn*:

$\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp (EVOL \varphi G T) \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
 $\langle proof \rangle$

lemma *pendulum-inv*:

$\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp (x' = f \ \& \ G) \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
 $\langle proof \rangle$

lemma *local-flow-pend*: *local-flow* f *UNIV UNIV* φ
 ⟨*proof*⟩

lemma *pendulum-flow*:
 $\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp (x' = f \ \& \ G) \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
 ⟨*proof*⟩

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 (φ)
 where $\varphi \ g \ t \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } g * t^{\wedge} 2 / 2 + s\$2 * t + s\$1 \text{ else } g * t + s\$2)$

— 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 $(x::\text{real}) \leq h$
 ⟨*proof*⟩

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

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>

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>

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))
 $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
<proof>

lemma *local-flow-ball*: *local-flow* (f g) *UNIV UNIV* (φ g)

<proof>

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))
 $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
<proof>

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, $\mathcal{2}$ 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>

lemma *local-lipschitz-temp-dyn*:

assumes $0 < (a::real)$

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

<proof>

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

<proof>

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>

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

assumes $a > 0$ **and** *Thyps*: $Tmin \leq T\ T \leq Tmax\ Tmax < (L::real)$

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

shows $L - Tmax \leq \exp(-(a * t)) * (L - T)$

and $L - \exp(-(a * t)) * (L - T) \leq Tmax$

and $Tmin \leq L - \exp(-(a * t)) * (L - T)$

<proof>

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

(LOOP

— control

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

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

$(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1) THEN (4 ::= (\lambda s.0)) ELSE skip));$
 — dynamics
 $(IF (\lambda s. s\$4 = 0) THEN (x' = f a 0 \ \& \ (\lambda s. s\$2 \leq - (\ln (Tmin/s\$3))/a))$
 $ELSE (x' = f a L \ \& \ (\lambda s. s\$2 \leq - (\ln ((L-Tmax)/(L-s\$3))/a)))$
 $INV (\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
 $[\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax]$
 $\langle proof \rangle$

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* (φk)
 $\langle proof \rangle$

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 $y \leq hmax \implies y - c_o * \tau \leq hmax$
 ⟨proof⟩

lemma *tank-flow*:

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

shows $[\lambda s. I hmin hmax s] \leq wp$

(*LOOP*

— control

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

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

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

— 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$

$[\lambda s. I hmin hmax s]$

⟨proof⟩

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*: $(\bigwedge x. (f \bullet) x = (g \bullet) x) \implies f = g$

⟨proof⟩

lemma *nd-fun-eq-iff*: $(f = g) = (\forall x. (f \bullet) x = (g \bullet) x)$

⟨proof⟩

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

definition $0 = \zeta^\bullet$

definition *star-nd-fun* $f = \text{qstar } f$ **for** $f::'a$ *nd-fun*

definition $f + g = ((f \bullet) \sqcup (g \bullet))^\bullet$

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

named-theorems *nd-fun-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*
<proof>

lemma *nd-fun-distr*[*nd-fun-ka*]: $(x + y) \cdot z = x \cdot z + y \cdot z$
and *nd-fun-distl*[*nd-fun-ka*]: $x \cdot (y + z) = x \cdot y + x \cdot z$ **for** $x::'a$ *nd-fun*
<proof>

lemma *nd-fun-plus-zero1*[*nd-fun-ka*]: $0 + x = x$
and *nd-fun-mult-zero1*[*nd-fun-ka*]: $0 \cdot x = 0$
and *nd-fun-mult-zero2*[*nd-fun-ka*]: $x \cdot 0 = 0$ **for** $x::'a$ *nd-fun*
<proof>

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

lemma *nd-star-one*[*nd-fun-ka*]: $1 + x \cdot x^\star \leq x^\star$
and *nd-star-unfoldl*[*nd-fun-ka*]: $z + x \cdot y \leq y \implies x^\star \cdot z \leq y$
and *nd-star-unfoldr*[*nd-fun-ka*]: $z + y \cdot x \leq y \implies z \cdot x^\star \leq y$ **for** $x::'a$ *nd-fun*
<proof>

instance
<proof>

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** $x :: 'a\ nd-fun$

<proof>

instance

<proof>

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* ($\lceil _ \rceil$)

and *Relation.relcomp* (**infixl** ; 75)

and *Range-Semiring.antirange-semiring-class.ars-r* (*r*)

and *antidomain-semiringl.ads-d* (*d*)

abbreviation *p2ndf* :: $'a\ pred \Rightarrow 'a\ nd-fun\ ((1 \lceil _ \rceil))$

where $\lceil Q \rceil \equiv (\lambda x :: 'a. \{s :: 'a. s = x \wedge Q\ s\})^\bullet$

lemma *p2ndf-simps*[*simp*]:

$$\begin{aligned}
[P] \leq [Q] &= (\forall s. P s \longrightarrow Q s) \\
([P] = [Q]) &= (\forall s. P s = Q s) \\
([P] \cdot [Q]) &= [\lambda s. P s \wedge Q s] \\
([P] + [Q]) &= [\lambda s. P s \vee Q s] \\
ad [P] &= [\lambda s. \neg P s] \\
d [P] &= [P] \quad [P] \leq \eta^\bullet \\
&\langle proof \rangle
\end{aligned}$$

Lemmas for verification condition generation

lemma *wp-nd-fun*: $wp F [P] = [\lambda s. \forall s'. s' \in ((F \bullet) s) \longrightarrow P s]$
 $\langle proof \rangle$

abbreviation *skip* :: 'a nd-fun
where *skip* $\equiv 1$

— Tests

lemma *wp-test[simp]*: $wp [P] [Q] = [\lambda s. P s \longrightarrow Q s]$
 $\langle proof \rangle$

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

lemma *wp-assign[simp]*: $wp (x ::= e) [Q] = [\lambda s. Q (\chi j. (((\$) s)(x := (e s))) j)]$
 $\langle proof \rangle$

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

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

lemma *le-wp-choice-iff*: $[P] \leq wp (X + Y) [Q] \longleftrightarrow [P] \leq wp X [Q] \wedge [P] \leq wp Y [Q]$
 $\langle proof \rangle$

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

— Conditional statement

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

— Finite iteration

abbreviation *loopi-sugar* :: 'a nd-fun \Rightarrow 'a pred \Rightarrow 'a nd-fun (**LOOP - INV** -

[64,64] 63)

where $LOOP\ R\ INV\ I \equiv aka-loopI\ R\ [I]$

lemma *change-loopI*: $LOOP\ X\ INV\ G = LOOP\ X\ INV\ I$

<proof>

lemma *wp-loopI*: $[P] \leq [I] \implies [I] \leq [Q] \implies [I] \leq wp\ R\ [I] \implies [P] \leq wp\ (LOOP\ R\ INV\ I)\ [Q]$

<proof>

lemma *wp-loopI-break*:

$[P] \leq wp\ Y\ [I] \implies [I] \leq wp\ X\ [I] \implies [I] \leq [Q] \implies [P] \leq wp\ (Y ; (LOOP\ X\ INV\ I))\ [Q]$

<proof>

6.6.2 Evolution commands

Verification by providing evolution

definition *g-evol* :: $((a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b\ pred \Rightarrow ('b \Rightarrow 'a\ set) \Rightarrow 'b\ nd-fun\ (EVOL)$

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 $wp\ (EVOL\ \varphi\ G\ U)\ [Q] = [\lambda s. \forall t \in U\ s. (\forall \tau \in down\ (U\ s)\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)]$

<proof>

Verification by providing solutions

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

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

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

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

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

<proof>

context *local-flow*

begin

lemma *wp-g-ode-subset*:

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

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

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

<proof>

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

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

<proof>

lemma *wp-g-ode-ivl*: $t \geq 0 \implies t \in T \implies wp (x' = (\lambda t. f) \ \& \ G \ on \ (\lambda s. \{0..t\}) \ S \ @ \ 0) \ [Q] =$
 $[\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))]$
<proof>

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

end

Verification with differential invariants

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

lemma *wp-g-orbital-guard*:
assumes $H = (\lambda s. G \ s \ \wedge \ Q \ s)$
shows $wp (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ [Q] = wp (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ [H]$
<proof>

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

lemma *wp-diff-inv[simp]*: $([I] \leq wp (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ [I]) = diff\ invariant$
 $I \ f \ U \ S \ t_0 \ G$
<proof>

lemma *diff-inv-guard-ignore*:
assumes $[I] \leq wp (x' = f \ \& \ (\lambda s. True) \ on \ U \ S \ @ \ t_0) \ [I]$
shows $[I] \leq wp (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ [I]$
<proof>

context *local-flow*
begin

lemma *wp-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 $diff\ invariant \ I \ (\lambda t. f) \ U \ S \ 0 \ (\lambda s. True) =$
 $([\lambda s. s \in S \longrightarrow I \ s] = wp (x' = (\lambda t. f) \ \& \ (\lambda s. True) \ on \ U \ S \ @ \ 0) \ [\lambda s. s \in S$
 $\longrightarrow I \ s])$
<proof>

lemma *diff-inv-eq-inv-set*:

diff-invariant $I (\lambda t. f) (\lambda s. T) S 0 (\lambda s. True) = (\forall s. I s \longrightarrow \gamma^\varphi s \subseteq \{s. I s\})$
 ⟨proof⟩

end

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

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

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

lemma *diff-solve-axiom1*:
assumes *local-flow* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$
shows $wp (x' = f \ \& \ G) \lceil Q \rceil =$
 $\lceil \lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s) \rceil$
 ⟨proof⟩

lemma *diff-solve-axiom2*:
fixes $c::a::\{\text{heine-borel}, \text{banach}\}$
shows $wp (x' = (\lambda s. c) \ \& \ G) \lceil Q \rceil =$
 $\lceil \lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (s + \tau *_R c)) \longrightarrow Q (s + t *_R c) \rceil$
 ⟨proof⟩

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 $\lceil P \rceil \leq wp (x' = f \ \& \ G) \lceil Q \rceil$
 ⟨proof⟩

lemma *diff-weak-axiom1*: $\eta^\bullet \leq wp (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \lceil G \rceil$
 ⟨proof⟩

lemma *diff-weak-axiom2*:
 $wp (x' = f \ \& \ G \ on \ T \ S \ @ \ t_0) \lceil Q \rceil = wp (x' = f \ \& \ G \ on \ T \ S \ @ \ t_0) \lceil \lambda s. G s \longrightarrow Q s \rceil$
 ⟨proof⟩

lemma *diff-weak-rule*:

assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *wp-g-orbit-IdD*:

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

lemma *diff-cut-axiom*:

assumes $wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil C \rceil = \eta^\bullet$
shows $wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil = wp(x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *diff-cut-rule*:

assumes $wp\text{-}C: \lceil P \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil C \rceil$
and $wp\text{-}Q: \lceil P \rceil \leq wp(x' = f \ \& \ (\lambda s. \ G \ s \ \wedge \ C \ s) \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$
shows $\lceil P \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$
 $\langle proof \rangle$

lemma *diff-inv-axiom1*:

assumes $G \ s \longrightarrow I \ s$ **and** *diff-invariant* $I(\lambda t. f) (\lambda s. \{t. t \geq 0\}) \text{ UNIV } 0 \ G$
shows $s \in ((wp(x' = f \ \& \ G) \lceil I \rceil)^\bullet) \ s$
 $\langle proof \rangle$

lemma *diff-inv-axiom2*:

assumes *picard-lindelof* $(\lambda t. f) \text{ UNIV UNIV } 0$
and $\bigwedge s. \{t::\text{real}. t \geq 0\} \subseteq \text{picard-lindelof.ex-ivl } (\lambda t. f) \text{ UNIV UNIV } 0 \ s$
and *diff-invariant* $I(\lambda t. f) (\lambda s. \{t::\text{real}. t \geq 0\}) \text{ UNIV } 0 \ G$
shows $wp(x' = f \ \& \ G) \lceil I \rceil = wp \lceil G \rceil \lceil I \rceil$
 $\langle proof \rangle$

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

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 $x' t = y t$ 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* :: $real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
where $\varphi t s \equiv (\chi i. \text{if } i = 1 \text{ then } s\$1 * \cos t + s\$2 * \sin t$
 $\text{else } -s\$1 * \sin t + s\$2 * \cos t)$

— Verified by providing dynamics.

lemma *pendulum-dyn*:
 $[\lambda s. r^2 = (s\$1)^2 + (s\$2)^2] \leq wp (EVOL \varphi G T) [\lambda s. r^2 = (s\$1)^2 + (s\$2)^2]$
<proof>

lemma *pendulum-inv*:
 $[\lambda s. r^2 = (s\$1)^2 + (s\$2)^2] \leq wp (x' = f \ \& \ G) [\lambda s. r^2 = (s\$1)^2 + (s\$2)^2]$
<proof>

lemma *local-flow-pend*: *local-flow f UNIV UNIV φ*
<proof>

lemma *pendulum-flow*:
 $[\lambda s. r^2 = (s\$1)^2 + (s\$2)^2] \leq wp (x' = f \ \& \ G) [\lambda s. r^2 = (s\$1)^2 + (s\$2)^2]$
<proof>

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* $:: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}^2 \Rightarrow \text{real}^2 (\varphi)$

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 $(x::\text{real}) \leq h$

<proof>

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*

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

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

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

$) [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$

<proof>

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>

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>

lemma *bouncing-ball-dyn*:

fixes $h::\text{real}$

assumes $g < 0$ and $h \geq 0$

shows $g < 0 \implies h \geq 0 \implies$

$[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq \text{wp}$

(*LOOP*

$((\text{EVOL } (\varphi \ g) (\lambda s. 0 \leq s\$1) \ T);$

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

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

$[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$

<proof>

lemma *local-flow-ball*: *local-flow* ($f\ g$) *UNIV UNIV* ($\varphi\ g$)
<proof>

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 ($\varrho ::= (\lambda s. - s\$2)$) ELSE skip))
*INV ($\lambda s. 0 \leq s\$1 \wedge \varrho * g * s\$1 = \varrho * g * h + s\$2 * s\2))*
 $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
<proof>

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 ($s\$4 = 1$) 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>

lemma *local-lipschitz-temp-dyn*:

assumes $0 < (a::real)$
shows *local-lipschitz UNIV UNIV* ($\lambda t::real. f\ a\ L$)

<proof>

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

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

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

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

assumes $a > 0$ **and** *Thyyps*: $T_{min} \leq T \ T \leq T_{max} \ T_{max} < (L::\text{real})$
and *thyyps*: $0 \leq t \ \forall \tau \in \{0..t\}. \tau \leq -(\ln ((L - T_{max}) / (L - T)) / a)$
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>

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

lemma *thermostat*:

assumes $a > 0$ **and** $0 < T_{min}$ **and** $T_{max} < L$
shows $[\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max} \wedge s\$4 = 0] \leq wp$
(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
 $(IF (\lambda s. s\$4 = 0) THEN (x' = f \ a \ 0 \ \& \ (\lambda s. s\$2 \leq -(\ln (T_{min}/s\$3))/a))$
 $ELSE (x' = f \ a \ L \ \& \ (\lambda s. s\$2 \leq -(\ln ((L - T_{max})/(L - s\$3))/a)))$
 $INV (\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max} \wedge (s\$4 = 0 \vee s\$4 = 1))$
 $[\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max}]$
<proof>

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 $h_{min} \leq h \leq h_{max}$. 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, \mathcal{J} is the level of water measured by the chronometer, and $\mathcal{4}$ states whether the pump is on ($s\mathcal{4} = 1$) or off ($s\mathcal{4} = 0$). We prove that the controller keeps the level of water between $hmin$ and $hmax$.

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

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

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

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

lemma *local-flow-tank*: *local-flow* ($f\ k$) *UNIV UNIV* ($\varphi\ k$)
 $\langle proof \rangle$

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 $y \leq hmax \implies y - c_o * \tau \leq hmax$
 $\langle proof \rangle$

lemma *tank-flow*:

assumes $0 < c_o$ **and** $c_o < c_i$
shows $[\lambda s.\ I\ hmin\ hmax\ s] \leq wp$
 $(LOOP$
 — control
 $((\mathcal{2} ::= (\lambda s. 0)); (\mathcal{3} ::= (\lambda s. s\mathcal{1})));$
 $(IF\ (\lambda s.\ s\mathcal{4} = 0 \wedge s\mathcal{3} \leq hmin + 1)\ THEN\ (\mathcal{4} ::= (\lambda s. 1))\ ELSE$
 $(IF\ (\lambda s.\ s\mathcal{4} = 1 \wedge s\mathcal{3} \geq hmax - 1)\ THEN\ (\mathcal{4} ::= (\lambda s. 0))\ ELSE\ skip));$
 — dynamics
 $(IF\ (\lambda s.\ s\mathcal{4} = 0)\ THEN\ (x' = f\ (c_i - c_o) \ \&\ (G\ hmax\ (c_i - c_o)))$
 $ELSE\ (x' = f\ (-c_o) \ \&\ (G\ hmin\ (-c_o))))\)\ INV\ I\ hmin\ hmax$
 $[\lambda s.\ I\ hmin\ hmax\ s]$
 $\langle proof \rangle$

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*: $\text{tt}\ p \leq \text{tt}\ p' \implies H\ p'\ x\ q \implies H\ p\ x\ q$
<proof>

lemma *H-consr*: $\text{tt}\ q' \leq \text{tt}\ q \implies H\ p\ x\ q' \implies H\ p\ x\ q$
<proof>

lemma *H-cons*: $\text{tt}\ p \leq \text{tt}\ p' \implies \text{tt}\ q' \leq \text{tt}\ q \implies H\ p'\ x\ q' \implies H\ p\ x\ q$
<proof>

lemma *H-skip*: $H\ p\ 1\ p$
<proof>

lemma *H-abort*: $H\ p\ 0\ q$
<proof>

lemma *H-seq*: $H p x r \implies H r y q \implies H p (x \cdot y) q$
 ⟨proof⟩

lemma *H-choice*: $H p x q \implies H p y q \implies H p (x + y) q$
 ⟨proof⟩

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

lemma *H-var*: $H p x q \iff \text{tt } p \cdot x \cdot n q = 0$
 ⟨proof⟩

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

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

definition *kat-while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - [64,64] 63*) **where**
while b do x = (tt b · x) · 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*: $H (\text{tt } p \cdot \text{tt } r) x q \implies H p (\text{tt } r \cdot x) q$
 ⟨proof⟩

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⟩

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

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

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⟩

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

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

<proof>

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

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

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

lemma *H-inv-mult*: $\text{tt } i = i \implies \text{tt } j = j \implies H \text{ i x i} \implies H j x j \implies H (i \cdot j) x (i \cdot j)$
<proof>

end

7.2 refinement KAT

Here we derive the laws of the refinement calculus.

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

begin

lemma *R1*: $H p (\text{Ref } p \ q) \ q$
<proof>

lemma *R2*: $H p \ x \ q \implies x \leq \text{Ref } p \ q$
<proof>

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>

lemma *R-skip*: $1 \leq \text{Ref } p \ p$
<proof>

lemma *R-zero-one*: $x \leq \text{Ref } 0 \ 1$
<proof>

lemma *R-one-zero*: $\text{Ref } 1 \ 0 = 0$
<proof>

lemma *R-abort*: $0 \leq \text{Ref } p \ q$
<proof>

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

<proof>

lemma *R-choice*: $(Ref\ p\ q) + (Ref\ p\ q) \leq Ref\ p\ q$
<proof>

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

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

lemma *R-star*: $(Ref\ i\ i)^* \leq Ref\ i\ i$
<proof>

lemma *R-loop*: *loop* $(Ref\ p\ p)$ *inv* $i \leq Ref\ p\ p$
<proof>

lemma *R-inv*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies Ref\ i\ i \leq Ref\ p\ q$
<proof>

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* $(\cup) (O) Id \{\}$ $(\subseteq) (\subset) \lambda x. Id \cap (-\ x)$
<proof>

interpretation *rel-kat*: *kat* $(\cup) (O) Id \{\}$ $(\subseteq) (\subset) rtrancl\ \lambda x. Id \cap (-\ x)$
<proof>

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* $(\cup) (O) Id \{\}$ $(\subseteq) (\subset) rtrancl\ (\lambda X. Id \cap -\ X)\ rel-R$

<proof>

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 $p2r-simps[simp]$:
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P s \longrightarrow Q s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P s = Q s)$
 $(\lceil P \rceil ; \lceil Q \rceil) = \lceil \lambda s. P s \wedge Q s \rceil$
 $(\lceil P \rceil \cup \lceil Q \rceil) = \lceil \lambda s. P s \vee Q s \rceil$
 $rel-tests.t \lceil P \rceil = \lceil P \rceil$
 $(- Id) \cup \lceil P \rceil = - \lceil \lambda s. \neg P s \rceil$
 $Id \cap (- \lceil P \rceil) = \lceil \lambda s. \neg P s \rceil$
<proof>

Lemmas for verification condition generation

lemma $RdL-is-rRKAT$: $(\forall x. \{(x,x)\}; R1 \subseteq \{(x,x)\}; R2) = (R1 \subseteq R2)$
<proof>

abbreviation $relHoare$ ($\{-\}-\{-\}$)
where $\{P\}X\{Q\} \equiv rel-kat.Hoare \lceil P \rceil X \lceil Q \rceil$

lemma $rel-kat-H$: $\{P\} X \{Q\} \longleftrightarrow (\forall s s'. P s \longrightarrow (s,s') \in X \longrightarrow Q s')$
<proof>

lemma $sH-skip[simp]$: $\{P\} skip \{Q\} \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
<proof>

lemma $H-skip$: $\{P\} skip \{P\}$
<proof>

lemma $sH-test[simp]$: $\{P\} \lceil R \rceil \{Q\} = (\forall s. P s \longrightarrow R s \longrightarrow Q s)$
<proof>

lemma $sH-abort[simp]$: $\{P\} abort \{Q\} \longleftrightarrow True$

$\langle proof \rangle$

lemma *H-abort*: $\{P\} \text{ abort } \{Q\}$
 $\langle proof \rangle$

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]*: $\{P\} (x ::= e) \{Q\} \longleftrightarrow (\forall s. P \ s \longrightarrow Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j))$
 $\langle proof \rangle$

lemma *H-assign*: $P = (\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j)) \Longrightarrow \{P\} (x ::= e) \{Q\}$
 $\langle proof \rangle$

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

lemma *sH-nondet-assign[simp]*:
 $\{P\} (x ::= ?) \{Q\} \longleftrightarrow (\forall s. P \ s \longrightarrow (\forall k. Q \ (\chi \ j. (((\$) \ s)(x := k)) \ j)))$
 $\langle proof \rangle$

lemma *H-nondet-assign*: $\{\lambda s. \forall k. P \ (\chi \ j. (((\$) \ s)(x := k)) \ j)\} (x ::= ?) \{P\}$
 $\langle proof \rangle$

lemma *H-seq*: $\{P\} \ X \ \{R\} \Longrightarrow \{R\} \ Y \ \{Q\} \Longrightarrow \{P\} \ X; Y \ \{Q\}$
 $\langle proof \rangle$

lemma *sH-seq*: $\{P\} \ X; Y \ \{Q\} = \{P\} \ X \ \{\lambda s. \forall s'. (s, s') \in Y \longrightarrow Q \ s'\}$
 $\langle proof \rangle$

lemma *H-assignl*:
assumes $\{K\} \ X \ \{Q\}$
and $\forall s. P \ s \longrightarrow K \ (\text{vec-lambda } (((\$) \ s)(x := e \ s)))$
shows $\{P\} (x ::= e); X \ \{Q\}$
 $\langle proof \rangle$

lemma *sH-choice*: $\{P\} \ X \cup Y \ \{Q\} \longleftrightarrow (\{P\} \ X \ \{Q\} \wedge \{P\} \ Y \ \{Q\})$
 $\langle proof \rangle$

lemma *H-choice*: $\{P\} \ X \ \{Q\} \Longrightarrow \{P\} \ Y \ \{Q\} \Longrightarrow \{P\} \ X \cup Y \ \{Q\}$
 $\langle proof \rangle$

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

lemma *sH-cond[simp]*:
 $\{P\} (\text{IF } B \ \text{THEN } X \ \text{ELSE } Y) \{Q\} \longleftrightarrow (\{\lambda s. P \ s \wedge B \ s\} \ X \ \{Q\} \wedge \{\lambda s. P \ s \wedge$

$\neg B s\} Y \{Q\}$
 $\langle proof \rangle$

lemma *H-cond*:

$\{\lambda s. P s \wedge B s\} X \{Q\} \Longrightarrow \{\lambda s. P s \wedge \neg B s\} Y \{Q\} \Longrightarrow \{P\}$ (*IF B THEN X ELSE Y*) $\{Q\}$
 $\langle proof \rangle$

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*: $\forall s. P s \longrightarrow I s \Longrightarrow \forall s. I s \wedge \neg B s \longrightarrow Q s \Longrightarrow \{\lambda s. I s \wedge B s\} X \{I\}$

$\Longrightarrow \{P\}$ (*WHILE B INV I DO X*) $\{Q\}$

$\langle proof \rangle$

lemma $\{\lambda s. P s \wedge B s\} X \{\lambda s. P s\} \Longrightarrow \{P\}$ (*WHILE B INV I DO X*) $\{\lambda s. P s \wedge \neg B s\}$

$\langle proof \rangle$

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*: $\{P\} X \{P\} \Longrightarrow \{P\}$ (*LOOP X INV I*) $\{P\}$

$\langle proof \rangle$

lemma *H-loopI*: $\{I\} X \{I\} \Longrightarrow [P] \subseteq [I] \Longrightarrow [I] \subseteq [Q] \Longrightarrow \{P\}$ (*LOOP X INV I*) $\{Q\}$

$\langle proof \rangle$

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

$\langle proof \rangle$

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 $\{P\} (\text{EVOL } \varphi G U) \{Q\}$

$\langle proof \rangle$

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

lemma *H-g-orbital*:

$P = (\lambda s. (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))) \implies$
 $\{P\} (x' = f \& G \text{ on } U S @ t_0) \{Q\}$
 $\langle \text{proof} \rangle$

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

$(\forall s. P s \longrightarrow (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t)))$
 $\langle \text{proof} \rangle$

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 $\{P\} (x' = (\lambda t. f) \& G \text{ on } U S @ 0) \{Q\} =$
 $(\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle \text{proof} \rangle$

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 $\{P\} (x' = (\lambda t. f) \& G \text{ on } U S @ 0) \{Q\}$
 $\langle \text{proof} \rangle$

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

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

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

$(\forall s \in S. P s \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle \text{proof} \rangle$

lemma *sH-orbit*: $\{P\} (\{(s, s') \mid s \leq s', s' \in \gamma^\varphi s\}) \{Q\} = (\forall s \in S. P s \longrightarrow (\forall t \in T. Q (\varphi t s)))$

$\langle \text{proof} \rangle$

end

— Verification with differential invariants

definition *g-ode-inv* :: (*real* \Rightarrow (*'a::banach*) \Rightarrow '*a*) \Rightarrow '*a pred* \Rightarrow (*'a* \Rightarrow *real set*) \Rightarrow '*a set* \Rightarrow
real \Rightarrow '*a pred* \Rightarrow '*a rel* ((*!x'* = - & - on - - @ - DINV -))
where (*x' = f & G on U S @ t0 DINV I*) = (*x' = f & G on U S @ t0*)

lemma *sH-g-orbital-guard*:
assumes *R* = ($\lambda s. G s \wedge Q s$)
shows $\{P\} (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \{Q\} = \{P\} (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \{R\}$
 $\{R\}$
 $\langle proof \rangle$

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

lemma *sH-diff-inv[simp]*: $\{I\} (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \{I\} = \text{diff-invariant } I \text{ f}$
 $U \ S \ t_0 \ G$
 $\langle proof \rangle$

lemma *H-g-ode-inv*: $\{I\} (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \{I\} \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow$
 $\lceil \lambda s. I s \wedge G s \rceil \leq \lceil Q \rceil \Longrightarrow \{P\} (x' = f \ \& \ G \ on \ U \ S \ @ \ t_0 \ \text{DINV } I) \{Q\}$
 $\langle proof \rangle$

7.4 Refinement Components

lemma *R-skip*: $(\forall s. P s \longrightarrow Q s) \Longrightarrow \text{skip} \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 $\langle proof \rangle$

lemma *R-abort*: $\text{abort} \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 $\langle proof \rangle$

lemma *R-seq*: $(\text{rel-R } \lceil P \rceil \lceil R \rceil) ; (\text{rel-R } \lceil R \rceil \lceil Q \rceil) \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 $\langle proof \rangle$

lemma *R-seq-law*: $X \leq \text{rel-R } \lceil P \rceil \lceil R \rceil \Longrightarrow Y \leq \text{rel-R } \lceil R \rceil \lceil Q \rceil \Longrightarrow X ; Y \leq \text{rel-R}$
 $\lceil P \rceil \lceil Q \rceil$
 $\langle proof \rangle$

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

— Nondeterministic Choice

lemma *R-choice*: $(\text{rel-R } \lceil P \rceil \lceil Q \rceil) \cup (\text{rel-R } \lceil P \rceil \lceil Q \rceil) \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
 $\langle proof \rangle$

lemma *R-choice-law*: $X \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil \Longrightarrow Y \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil \Longrightarrow X \cup Y \leq$
 $\text{rel-R } \lceil P \rceil \lceil Q \rceil$

<proof>

lemma *R-choice-mono*: $P' \subseteq P \implies Q' \subseteq Q \implies P' \cup Q' \subseteq P \cup Q$
<proof>

lemma *R-assign*: $(x ::= e) \leq \text{rel-R } [\lambda s. P (\chi j. (((\$) s)(x := e s)) j)] [P]$
<proof>

lemma *R-assign-law*:
 $(\forall s. P s \longrightarrow Q (\chi j. (((\$) s)(x := (e s))) j)) \implies (x ::= e) \leq \text{rel-R } [P] [Q]$
<proof>

lemma *R-assignl*: $P = (\lambda s. R (\chi j. (((\$) s)(x := e s)) j)) \implies$
 $(x ::= e) ; \text{rel-R } [R] [Q] \leq \text{rel-R } [P] [Q]$
<proof>

lemma *R-assignr*: $R = (\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)) \implies$
 $\text{rel-R } [P] [R]; (x ::= e) \leq \text{rel-R } [P] [Q]$
<proof>

lemma $(x ::= e) ; \text{rel-R } [Q] [Q] \leq \text{rel-R } [(\lambda s. Q (\chi j. (((\$) s)(x := e s)) j))] [Q]$
<proof>

lemma $\text{rel-R } [Q] [(\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)]; (x ::= e) \leq \text{rel-R } [Q] [Q]$
<proof>

lemma *R-nondet-assign*: $(x ::= ?) \leq \text{rel-R } [\lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j)] [P]$
<proof>

lemma *R-nondet-assign-law*:
 $(\forall s. P s \longrightarrow (\forall k. Q (\chi j. (((\$) s)(x := k)) j))) \implies (x ::= ?) \leq \text{rel-R } [P] [Q]$
<proof>

lemma *R-cond*:
 $(\text{IF } B \text{ THEN } \text{rel-R } [\lambda s. B s \wedge P s] [Q] \text{ ELSE } \text{rel-R } [\lambda s. \neg B s \wedge P s] [Q]) \leq$
 $\text{rel-R } [P] [Q]$
<proof>

lemma *R-cond-mono*: $X \leq X' \implies Y \leq Y' \implies (\text{IF } P \text{ THEN } X \text{ ELSE } Y) \leq \text{IF } P \text{ THEN } X' \text{ ELSE } Y'$
<proof>

lemma *R-cond-law*: $X \leq \text{rel-R } [\lambda s. B s \wedge P s] [Q] \implies Y \leq \text{rel-R } [\lambda s. \neg B s \wedge P s] [Q] \implies$
 $(\text{IF } B \text{ THEN } X \text{ ELSE } Y) \leq \text{rel-R } [P] [Q]$
<proof>

lemma *R-while*: $K = (\lambda s. P s \wedge \neg B s) \implies$
 $WHILE B INV I DO (rel-R [\lambda s. P s \wedge B s] [P]) \leq rel-R [P] [K]$
 ⟨proof⟩

lemma *R-whileI*:
 $X \leq rel-R [I] [I] \implies [P] \leq [\lambda s. I s \wedge B s] \implies [\lambda s. I s \wedge \neg B s] \leq [Q] \implies$
 $WHILE B INV I DO X \leq rel-R [P] [Q]$
 ⟨proof⟩

lemma *R-while-mono*: $X \leq X' \implies (WHILE P INV I DO X) \subseteq WHILE P INV I DO X'$
 ⟨proof⟩

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
 $(WHILE B INV I DO X) \leq rel-R [P] [Q]$
 ⟨proof⟩

lemma *R-loop*: $LOOP rel-R [P] [P] INV I \leq rel-R [P] [P]$
 ⟨proof⟩

lemma *R-loopI*:
 $X \leq rel-R [I] [I] \implies [P] \leq [I] \implies [I] \leq [Q] \implies LOOP X INV I \leq rel-R$
 $[P] [Q]$
 ⟨proof⟩

lemma *R-loop-mono*: $X \leq X' \implies LOOP X INV I \subseteq LOOP X' INV I$
 ⟨proof⟩

lemma *R-g-evol*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(EVOL \varphi G U) \leq rel-R [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow$
 $P (\varphi t s)] [P]$
 ⟨proof⟩

lemma *R-g-evol-law*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall s. P s \longrightarrow (\forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\implies (EVOL \varphi G U) \leq rel-R [P] [Q]$
 ⟨proof⟩

lemma *R-g-evoll*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \implies$
 $(EVOL \varphi G U) ; rel-R [R] [Q] \leq rel-R [P] [Q]$
 ⟨proof⟩

lemma *R-g-evolr*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$

shows $R = (\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \Longrightarrow$
 $\text{rel-R } [P] [R]; (\text{EVOL } \varphi G U) \leq \text{rel-R } [P] [Q]$
 $\langle \text{proof} \rangle$

lemma

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

shows $\text{EVOL } \varphi G U ; \text{rel-R } [Q] [Q] \leq \text{rel-R } [\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)] [Q]$
 $\langle \text{proof} \rangle$

lemma

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

shows $\text{rel-R } [Q] [\lambda s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)];$
 $\text{EVOL } \varphi G U \leq \text{rel-R } [Q] [Q]$
 $\langle \text{proof} \rangle$

context *local-flow*

begin

lemma *R-g-ode-subset:*

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

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

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

$\langle \text{proof} \rangle$

lemma *R-g-ode-rule-subset:*

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

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

\Longrightarrow

$(x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) \leq \text{rel-R } [P] [Q]$

$\langle \text{proof} \rangle$

lemma *R-g-odel-subset:*

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

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 $(x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) ; \text{rel-R } [R] [Q] \leq \text{rel-R } [P] [Q]$

$\langle \text{proof} \rangle$

lemma *R-g-oder-subset:*

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

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 $\text{rel-R } [P] [R]; (x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) \leq \text{rel-R } [P] [Q]$

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

lemma *R-g-ode-law:* $(\forall s \in S. P s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q$

$(\varphi t s)) \implies$
 $(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) \ S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
 $\langle \text{proof} \rangle$

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow R (\varphi \ t \ s)) \implies$
 $(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) \ S \ @ \ 0) ; \text{rel-R } [R] \ [Q] \leq \text{rel-R } [P] \ [Q]$
 $\langle \text{proof} \rangle$

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)) \implies$
 $\text{rel-R } [P] \ [R]; (x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. T) \ S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
 $\langle \text{proof} \rangle$

lemma *R-g-ode-ivl*:
 $t \geq 0 \implies t \in T \implies (\forall s \in S. P \ s \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow$
 $Q (\varphi \ t \ s))) \implies$
 $(x' = (\lambda t. f) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \ S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
 $\langle \text{proof} \rangle$

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $I \ f \ T \ S \ t_0 \ G \implies [P] \leq [I] \implies [\lambda s. I \ s \ \wedge \ G$
 $s] \leq [Q] \implies$
 $(x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0 \ \text{DINV } I) \leq \text{rel-R } [P] \ [Q]$
 $\langle \text{proof} \rangle$

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* $f \ \text{UNIV} \ \text{UNIV} \ \varphi$

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 $\{P\} (x' = f \ \& \ G) \{Q\}$

$\langle \text{proof} \rangle$

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 $\{P\} (x' = (\lambda s. c) \ \& \ G) \{Q\}$

<proof>

lemma *diff-weak-rule*:

assumes $\lceil G \rceil \leq \lceil Q \rceil$

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

<proof>

lemma *diff-cut-rule*:

assumes *wp-C:rel-kat.Hoare* $\lceil P \rceil (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil C \rceil$

and *wp-Q:rel-kat.Hoare* $\lceil P \rceil (x' = f \ \& \ (\lambda \ s. \ G \ s \ \wedge \ C \ s) \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$

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

<proof>

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 $\{P\} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \{Q\}$

<proof>

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-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** $P::'a \ \text{pred}$

= (*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 ::  $real \Rightarrow real^2 \Rightarrow real^2$  ( $\varphi$ )
  where  $\varphi \tau s \equiv (\chi i. \text{if } i = 1 \text{ then } s\$1 \cdot \cos \tau + s\$2 \cdot \sin \tau$ 
   $\text{else } -s\$1 \cdot \sin \tau + s\$2 \cdot \cos \tau)$ 

```

— Verified with annotated dynamics

```

lemma pendulum-dyn:  $\{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} \text{EVOL } \varphi \ G \ T \ \{\lambda s. r^2 =$ 
 $(s \$ 1)^2 + (s \$ 2)^2\}$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma pendulum-inv:  $\{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} (x' = f \ \& \ G) \ \{\lambda s. r^2 = (s$ 
 $\$ 1)^2 + (s \$ 2)^2\}$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma local-flow-pend: local-flow f UNIV UNIV  $\varphi$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma pendulum-flow:  $\{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} (x' = f \ \& \ G) \ \{\lambda s. r^2 = (s$ 
 $\$ 1)^2 + (s \$ 2)^2\}$ 
   $\langle \text{proof} \rangle$ 

```

```

no-notation fpend (f)
  and pend-flow ( $\varphi$ )

```

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 $ball-flow :: real \Rightarrow real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
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>

lemma *fball-invariant*:

fixes $g h :: real$
defines *div*: $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows *diff-invariant* $I (\lambda t. f g) (\lambda s. UNIV) UNIV 0 G$
<proof>

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies$

$\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
 $((x' = f g \ \& \ (\lambda s. s\$1 \geq 0) \ \text{DINV } (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2 = 0));$
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
 $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
<proof>

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$
shows $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
and $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
<proof>

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
shows $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$
 $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs = ?rhs*)
<proof>

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 ($\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
<proof>

lemma *local-flow-ball*: *local-flow* (*f g*) *UNIV UNIV* (φ *g*)
<proof>

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 ($\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
) $\{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
<proof>

lemma *R-bb-assign*: $g < (0::real) \implies 0 \leq h \implies$
 $2 ::= (\lambda s. - s\$2) \leq \text{rel-R}$
 $[\lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
 $[\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
<proof>

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 ($\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$)
<proof>

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 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 $U = L \geq 0$ 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 ($s4 = 1$) or off ($s4 = 0$). 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 $f a L s \equiv (\chi i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } -a * (s1 - L) \text{ else } 0))$

abbreviation *therm-guard* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow bool (G)$
where $G Tmin Tmax a L s \equiv (s2 \leq -(\ln((L - (\text{if } L = 0 \text{ then } Tmin \text{ else } Tmax)) / (L - s3))) / a)$

abbreviation *therm-loop-inv* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool (I)$
where $I Tmin Tmax s \equiv Tmin \leq s1 \wedge s1 \leq Tmax \wedge (s4 = 0 \vee s4 = 1)$

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

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \implies \|f a L s_1 - f a L s_2\| = |a| * |s_1 - s_2|$
 $\langle proof \rangle$

lemma *local-lipschitz-therm-dyn*:
assumes $0 < (a::real)$
shows *local-lipschitz UNIV UNIV* $(\lambda t::real. f a L)$
 $\langle proof \rangle$

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

lemma *therm-dyn-down-real-arith*:
assumes $a > 0$ **and** *Thyps*: $0 < Tmin \ Tmin \leq T \ T \leq Tmax$
and *thyps*: $0 \leq (\tau::real) \ \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(Tmin / T) / a)$
shows $Tmin \leq \exp(-a * \tau) * T$ **and** $\exp(-a * \tau) * T \leq Tmax$
 $\langle proof \rangle$

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

lemmas $H\text{-}g\text{-ode}\text{-therm} = \text{local-flow.sH-g-ode-ivl}[OF \text{local-flow-therm} - UNIV\text{-}I]$

lemma *thermostat-flow*:

assumes $0 < a$ **and** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$

shows $\{I \ T_{min} \ T_{max}\}$

(*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}*)

$\{I \ T_{min} \ T_{max}\}$

<proof>

lemma *R-therm-dyn-down*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$

shows *rel-R* [$\lambda s. s\$4 = 0 \wedge I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I T_{min} T_{max}*] \geq

($x' = (\lambda t. f \ a \ 0)$ & $G \ T_{min} \ T_{max} \ a \ 0$ on ($\lambda s. \{0..\tau\}$) *UNIV @ 0*)

<proof>

lemma *R-therm-dyn-up*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$

shows *rel-R* [$\lambda s. s\$4 \neq 0 \wedge I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I T_{min} T_{max}*] \geq

($x' = (\lambda t. f \ a \ L)$ & $G \ T_{min} \ T_{max} \ a \ L$ on ($\lambda s. \{0..\tau\}$) *UNIV @ 0*)

<proof>

lemma *R-therm-dyn*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$

shows *rel-R* [$\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\1] [*I T_{min} T_{max}*] \geq

(*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*))

<proof>

lemma *R-therm-assign1*: *rel-R* [*I T_{min} T_{max}*] [$\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0$] \geq ($2 ::= (\lambda s. 0)$)

<proof>

lemma *R-therm-assign2*:

rel-R $[\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq (\beta ::= (\lambda s. s\$1))$
<proof>

lemma *R-therm-ctrl*:

rel-R $[I \ Tmin \ Tmax] [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$
 $(2 ::= (\lambda s. 0));$
 $(3 ::= (\lambda s. s\$1));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1) THEN$
 $(4 ::= (\lambda s. 1))$
 $ELSE IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1) THEN$
 $(4 ::= (\lambda s. 0))$
 $ELSE skip)$
<proof>

lemma *R-therm-loop*: *rel-R* $[I \ Tmin \ Tmax] [I \ Tmin \ Tmax] \geq$

(LOOP
rel-R $[I \ Tmin \ Tmax] [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1];$
rel-R $[\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] [I \ Tmin \ Tmax]$
INV $I \ Tmin \ Tmax)$
<proof>

lemma *R-thermostat-flow*:

assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$

shows *rel-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 $I \ Tmin \ Tmax)$
<proof>

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

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

— Verified with the flow

lemma *local-flow-tank*: *local-flow* ($f\ k$) UNIV UNIV ($\varphi\ k$)
 $\langle proof \rangle$

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 $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$
 $\langle proof \rangle$

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 $\{I \text{ hmin hmax}\}$
 (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 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ \text{hmax} (c_i - c_o) \ \text{on} \ (\lambda s. \{0.. \tau\}))$ UNIV @ 0
 ELSE $(x' = (\lambda t. f (-c_o)) \ \& \ G \ \text{hmin} (-c_o) \ \text{on} \ (\lambda s. \{0.. \tau\}))$ UNIV @ 0))
 INV I hmin hmax)
 $\{I \text{ hmin hmax}\}$
 <proof>

lemma *tank-diff-inv*:

$0 \leq \tau \implies \text{diff-invariant } (dI \ \text{hmin hmax } k) \ (\lambda t. f \ k) \ (\lambda s. \{0.. \tau\}) \ \text{UNIV } 0 \ \text{Guard}$
 <proof>

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>

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>

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
 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ \text{hmax} (c_i - c_o) \ \text{on} \ (\lambda s. \{0.. \tau\}))$ UNIV @ 0 DINV
 $(dI \ \text{hmin hmax} (c_i - c_o))$)
 ELSE
 $(x' = (\lambda t. f (-c_o)) \ \& \ G \ \text{hmin} (-c_o) \ \text{on} \ (\lambda s. \{0.. \tau\}))$ UNIV @ 0 DINV $(dI$
 $\text{hmin hmax} (-c_o))$)
 INV I hmin hmax)
 $\{I \text{ hmin hmax}\}$
 <proof>

abbreviation *tank-ctrl* :: *real* \Rightarrow *real* \Rightarrow (*real*⁴) *rel* (*ctrl*)
where *ctrl* *hmin* *hmax* \equiv ((*?* ::=($\lambda s.0$));(*?* ::=($\lambda s. s\$1$)));
(*IF* ($\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$) *THEN* (*?* ::= ($\lambda s.1$)) *ELSE*
(*IF* ($\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$) *THEN* (*?* ::= ($\lambda s.0$)) *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* τ) *INV* (*I* *hmin* *hmax*)

lemma *R-tank-inv*:

assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *rel-R* [*I* *hmin* *hmax*] [*I* *hmin* *hmax*] \geq
(*LOOP*
— control
((*?* ::=($\lambda s.0$));(*?* ::=($\lambda s. s\$1$)));
(*IF* ($\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$) *THEN* (*?* ::= ($\lambda s.1$)) *ELSE*
(*IF* ($\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$) *THEN* (*?* ::= ($\lambda s.0$)) *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*)
<*proof*>

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 = ( $\lambda x. \text{if } ((f \bullet) x = \{\}) \text{ then } \{x\} \text{ else } \{\}$ )•

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
   $x :: 'a \text{ nd-fun}$ 
   $\langle \text{proof} \rangle$ 

instance
   $\langle \text{proof} \rangle$ 

end

instantiation nd-fun :: (type) rkat
begin

definition Ref-nd-fun P Q  $\equiv (\lambda s. \bigcup \{(f \bullet) s \mid f. \text{Hoare } P f Q\})$ •

instance
   $\langle \text{proof} \rangle$ 

end

```

7.7.1 Regular programs

Lemmas for manipulation of predicates in the relational model

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

```

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

```

```

definition p2ndf :: 'a pred  $\Rightarrow$  'a nd-fun (( $\lceil \_ \rceil$ ))
  where  $\lceil Q \rceil \equiv (\lambda x :: 'a. \{s :: 'a. s = x \wedge Q s\})$ •

```

lemma *p2ndf-simps*[simp]:
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P\ s \longrightarrow Q\ s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P\ s = Q\ s)$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P\ s \wedge Q\ s \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P\ s \vee Q\ s \rceil$
 $\text{tt } \lceil P \rceil = \lceil P \rceil$
 $n\ \lceil P \rceil = \lceil \lambda s. \neg P\ s \rceil$
 $\langle \text{proof} \rangle$

Lemmas for verification condition generation

abbreviation *ndfunHoare* ($\{-\}-\{-\}$)
where $\{P\}X\{Q\} \equiv \text{Hoare } \lceil P \rceil\ X\ \lceil Q \rceil$

lemma *ndfun-kat-H*: $\{P\} X \{Q\} \longleftrightarrow (\forall s\ s'. P\ s \longrightarrow s' \in (X\bullet)\ s \longrightarrow Q\ s')$
 $\langle \text{proof} \rangle$

abbreviation *skip* $\equiv (1::'a\ \text{nd-fun})$

lemma *sH-skip*[simp]: $\{P\}\ \text{skip}\ \{Q\} \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *H-skip*: $\{P\}\ \text{skip}\ \{P\}$
 $\langle \text{proof} \rangle$

lemma *sH-test*[simp]: $\{P\}\ \lceil R \rceil\ \{Q\} = (\forall s. P\ s \longrightarrow R\ s \longrightarrow Q\ s)$
 $\langle \text{proof} \rangle$

abbreviation *abort* $\equiv (0::'a\ \text{nd-fun})$

lemma *sH-abort*[simp]: $\{P\}\ \text{abort}\ \{Q\} \longleftrightarrow \text{True}$
 $\langle \text{proof} \rangle$

lemma *H-abort*: $\{P\}\ \text{abort}\ \{Q\}$
 $\langle \text{proof} \rangle$

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

lemma *sH-assign*[simp]: $\{P\}\ (x ::= e)\ \{Q\} \longleftrightarrow (\forall s. P\ s \longrightarrow Q\ (\chi\ j. (((\$)\ s)(x := (e\ s)))\ j))$
 $\langle \text{proof} \rangle$

lemma *H-assign*: $P = (\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow \{P\}\ (x ::= e)\ \{Q\}$
 $\langle \text{proof} \rangle$

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

lemma *sH-nondet-assign*[simp]:

$$\{P\} (x ::= ?) \{Q\} \longleftrightarrow (\forall s. P s \longrightarrow (\forall k. Q (\chi j. (((\$) s)(x := k)) j)))$$

<proof>

lemma *H-nondet-assign*: $\{\lambda s. \forall k. P (\chi j. (((\$) s)(x := k)) j)\} (x ::= ?) \{P\}$
<proof>

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

lemma *H-seq*: $\{P\} X \{R\} \Longrightarrow \{R\} Y \{Q\} \Longrightarrow \{P\} X;Y \{Q\}$
<proof>

lemma *sH-seq*: $\{P\} X;Y \{Q\} = \{P\} X \{\lambda s. \forall s'. s' \in (Y \bullet) s \longrightarrow Q s'\}$
<proof>

lemma *H-assignl*:
assumes $\{K\} X \{Q\}$
and $\forall s. P s \longrightarrow K (\text{vec-lambda } (((\$) s)(x := e s)))$
shows $\{P\} (x ::= e);X \{Q\}$
<proof>

lemma *sH-choice*: $\{P\} X + Y \{Q\} \longleftrightarrow (\{P\} X \{Q\} \wedge \{P\} Y \{Q\})$
<proof>

lemma *H-choice*: $\{P\} X \{Q\} \Longrightarrow \{P\} Y \{Q\} \Longrightarrow \{P\} X + Y \{Q\}$
<proof>

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

lemma *sH-cond[simp]*:
 $\{P\} (IF B THEN X ELSE Y) \{Q\} \longleftrightarrow (\{\lambda s. P s \wedge B s\} X \{Q\} \wedge \{\lambda s. P s \wedge \neg B s\} Y \{Q\})$
<proof>

lemma *H-cond*:
 $\{\lambda s. P s \wedge B s\} X \{Q\} \Longrightarrow \{\lambda s. P s \wedge \neg B s\} Y \{Q\} \Longrightarrow \{P\} (IF B THEN X ELSE Y) \{Q\}$
<proof>

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

lemma *sH-whileI*: $\forall s. P s \longrightarrow I s \Longrightarrow \forall s. I s \wedge \neg B s \longrightarrow Q s \Longrightarrow \{\lambda s. I s \wedge B s\} X \{I\}$
 $\Longrightarrow \{P\} (WHILE B INV I DO X) \{Q\}$
<proof>

lemma $\{\lambda s. P s \wedge B s\} X \{\lambda s. P s\} \Longrightarrow \{P\}$ (*WHILE B INV I DO X*) $\{\lambda s. P s \wedge \neg B s\}$
 $\langle proof \rangle$

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

lemma *H-loop*: $\{P\} X \{P\} \Longrightarrow \{P\}$ (*LOOP X INV I*) $\{P\}$
 $\langle proof \rangle$

lemma *H-loopI*: $\{I\} X \{I\} \Longrightarrow [P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow \{P\}$ (*LOOP X*
INV I) $\{Q\}$
 $\langle proof \rangle$

7.7.2 Evolution commands

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

lemma *sH-g-evol[simp]*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $\{P\} (EVOL \varphi G U) \{Q\} = (\forall s. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
 $\langle proof \rangle$

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 $\{P\} (EVOL \varphi G U) \{Q\}$
 $\langle proof \rangle$

definition *g-ode* :: $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a$ *pred* $\Rightarrow ('a \Rightarrow real$ *set*) $\Rightarrow 'a$ *set*
 \Rightarrow
 $real \Rightarrow 'a$ *nd-fun* $((1x' = - \& - \text{on } - - @ -))$
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 *H-g-orbital*:
 $P = (\lambda s. (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t))) \Longrightarrow$
 $\{P\} (x' = f \& G \text{ on } U S @ t_0) \{Q\}$
 $\langle proof \rangle$

lemma *sH-g-orbital*: $\{P\} (x' = f \& G \text{ on } U S @ t_0) \{Q\} =$
 $(\forall s. P s \longrightarrow (\forall X \in \text{ivp-sols } f U S t_0 s. \forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (X \tau)) \longrightarrow Q (X t)))$
 $\langle proof \rangle$

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>

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

<proof>

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

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

<proof>

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

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

<proof>

lemma *sH-orbit*: $\{P\} \ \gamma^\varphi \bullet \ \{Q\} = (\forall s \in S. P s \longrightarrow (\forall t \in T. Q (\varphi \ t \ s)))$

<proof>

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 $\{P\} (x' = f \ \& \ G \ \text{on } U S \ @ \ t_0) \ \{Q\} = \{P\} (x' = f \ \& \ G \ \text{on } U S \ @ \ t_0) \ \{R\}$

<proof>

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 $\{P\} (x' = f \ \& \ G \ \text{on } U S \ @ \ t_0) \ \{Q\}$

<proof>

lemma *sH-diff-inv[simp]*: $\{I\} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \{I\} = \text{diff-invariant } I \text{ f}$
 $U \ S \ t_0 \ G$
 $\langle \text{proof} \rangle$

lemma *H-g-ode-inv*: $\{I\} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \{I\} \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow$
 $\lceil \lambda s. I \ s \ \wedge \ G \ s \rceil \leq \lceil Q \rceil \Longrightarrow \{P\} (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0 \ \text{DINV } I) \{Q\}$
 $\langle \text{proof} \rangle$

7.8 Refinement Components

lemma *R-skip*: $(\forall s. P \ s \longrightarrow Q \ s) \Longrightarrow 1 \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-abort*: $\text{abort} \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-seq*: $(\text{Ref } \lceil P \rceil \lceil R \rceil) ; (\text{Ref } \lceil R \rceil \lceil Q \rceil) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-seq-law*: $X \leq \text{Ref } \lceil P \rceil \lceil R \rceil \Longrightarrow Y \leq \text{Ref } \lceil R \rceil \lceil Q \rceil \Longrightarrow X ; Y \leq \text{Ref } \lceil P \rceil$
 $\lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemmas *R-seq-mono = mult-isol-var*

— Nondeterministic Choice

lemma *R-choice*: $(\text{Ref } \lceil P \rceil \lceil Q \rceil) + (\text{Ref } \lceil P \rceil \lceil Q \rceil) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-choice-law*: $X \leq \text{Ref } \lceil P \rceil \lceil Q \rceil \Longrightarrow Y \leq \text{Ref } \lceil P \rceil \lceil Q \rceil \Longrightarrow X + Y \leq$
 $\text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-choice-mono*: $P' \leq P \Longrightarrow Q' \leq Q \Longrightarrow P' + Q' \leq P + Q$
 $\langle \text{proof} \rangle$

lemma *R-assign*: $(x ::= e) \leq \text{Ref } \lceil \lambda s. P \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j) \rceil \lceil P \rceil$
 $\langle \text{proof} \rangle$

lemma *R-assign-law*:
 $(\forall s. P \ s \longrightarrow Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j)) \Longrightarrow (x ::= e) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-assignl*:
 $P = (\lambda s. R \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)) \Longrightarrow (x ::= e) ; \text{Ref } \lceil R \rceil \lceil Q \rceil \leq \text{Ref } \lceil P \rceil$
 $\lceil Q \rceil$
 $\langle \text{proof} \rangle$

lemma *R-assignr*:

$$R = (\lambda s. Q (\chi j. (((\$) s)(x := e s) j))) \implies \text{Ref } [P] [R]; (x ::= e) \leq \text{Ref } [P] [Q]$$

⟨proof⟩

lemma $(x ::= e) ; \text{Ref } [Q] [Q] \leq \text{Ref } [(\lambda s. Q (\chi j. (((\$) s)(x := e s) j))] [Q]$

⟨proof⟩

lemma $\text{Ref } [Q] [(\lambda s. Q (\chi j. (((\$) s)(x := e s) j))]; (x ::= e) \leq \text{Ref } [Q] [Q]$

⟨proof⟩

lemma *R-nondet-assign*: $(x ::= ?) \leq \text{Ref } [\lambda s. \forall k. P (\chi j. (((\$) s)(x := k) j))] [P]$

⟨proof⟩

lemma *R-nondet-assign-law*:

$$(\forall s. P s \longrightarrow (\forall k. Q (\chi j. (((\$) s)(x := k) j)))) \implies (x ::= ?) \leq \text{Ref } [P] [Q]$$

⟨proof⟩

lemma *R-cond*:

$$(IF B THEN \text{Ref } [\lambda s. B s \wedge P s] [Q] ELSE \text{Ref } [\lambda s. \neg B s \wedge P s] [Q]) \leq \text{Ref } [P] [Q]$$

⟨proof⟩

lemma *R-cond-mono*: $X \leq X' \implies Y \leq Y' \implies (IF P THEN X ELSE Y) \leq IF P THEN X' ELSE Y'$

⟨proof⟩

lemma *R-cond-law*: $X \leq \text{Ref } [\lambda s. B s \wedge P s] [Q] \implies Y \leq \text{Ref } [\lambda s. \neg B s \wedge P s] [Q] \implies (IF B THEN X ELSE Y) \leq \text{Ref } [P] [Q]$

⟨proof⟩

lemma *R-while*: $K = (\lambda s. P s \wedge \neg B s) \implies$

$$WHILE B INV I DO (\text{Ref } [\lambda s. P s \wedge B s] [P]) \leq \text{Ref } [P] [K]$$

⟨proof⟩

lemma *R-whileI*:

$$X \leq \text{Ref } [I] [I] \implies [P] \leq [\lambda s. I s \wedge B s] \implies [\lambda s. I s \wedge \neg B s] \leq [Q] \implies$$

$$WHILE B INV I DO X \leq \text{Ref } [P] [Q]$$

⟨proof⟩

lemma *R-while-mono*: $X \leq X' \implies (WHILE P INV I DO X) \leq WHILE P INV I DO X'$

⟨proof⟩

lemma *R-while-law*: $X \leq \text{Ref } [\lambda s. P s \wedge B s] [P] \implies Q = (\lambda s. P s \wedge \neg B s) \implies$

$$(WHILE B INV I DO X) \leq \text{Ref } [P] [Q]$$

<proof>

lemma *R-loop*: $X \leq \text{Ref } \lceil I \rceil \lceil I \rceil \implies \lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \lceil Q \rceil \implies \text{LOOP } X$
 $\text{INV } I \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
<proof>

lemma *R-loopI*: $X \leq \text{Ref } \lceil I \rceil \lceil I \rceil \implies \lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \lceil Q \rceil \implies \text{LOOP } X$
 $\text{INV } I \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
<proof>

lemma *R-loop-mono*: $X \leq X' \implies \text{LOOP } X \text{ INV } I \leq \text{LOOP } X' \text{ INV } I$
<proof>

lemma *R-g-evol*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $(\text{EVOL } \varphi \ G \ U) \leq \text{Ref } \lceil \lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow$
 $P (\varphi \ t \ s) \rceil \lceil P \rceil$
<proof>

lemma *R-g-evol-law*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall s. P \ s \longrightarrow (\forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)))$
 \implies
 $(\text{EVOL } \varphi \ G \ U) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
<proof>

lemma *R-g-evoll*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow R (\varphi \ t \ s)) \implies$
 $(\text{EVOL } \varphi \ G \ U) ; \text{Ref } \lceil R \rceil \lceil Q \rceil \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
<proof>

lemma *R-g-evolr*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)) \implies$
 $\text{Ref } \lceil P \rceil \lceil R \rceil ; (\text{EVOL } \varphi \ G \ U) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$
<proof>

lemma

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{EVOL } \varphi \ G \ U ; \text{Ref } \lceil Q \rceil \lceil Q \rceil \leq$
 $\text{Ref } \lceil \lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s) \rceil \lceil Q \rceil$
<proof>

lemma

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{Ref } \lceil Q \rceil \lceil \lambda s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s) \rceil ;$
 $\text{EVOL } \varphi \ G \ U \leq \text{Ref } \lceil Q \rceil \lceil Q \rceil$
<proof>

context *local-flow*

begin

lemma *R-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 $(x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) \leq$

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

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 $(x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) ; \text{Ref } [R] \ [Q] \leq \text{Ref } [P] \ [Q]$

$\langle \text{proof} \rangle$

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 $\text{Ref } [P] \ [R]; (x' = (\lambda t. f) \ \& \ G \ \text{on } U S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

lemma *R-g-ode-law*: $(\forall s \in S. P s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies$

$(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

$\langle \text{proof} \rangle$

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \implies$

$(x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) ; \text{Ref } [R] \ [Q] \leq \text{Ref } [P] \ [Q]$

$\langle \text{proof} \rangle$

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \implies$

$\text{Ref } [P] \ [R]; (x' = (\lambda t. f) \ \& \ G \ \text{on } (\lambda s. T) S \ @ \ 0) \leq \text{Ref } [P] \ [Q]$

$\langle \text{proof} \rangle$

lemma *R-g-ode-ivl*:

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

$(x' = (\lambda t. f) \ \& \ G \ \text{on} \ (\lambda s. \{0..t\}) \ S \ @ \ 0) \leq \text{Ref} \ [P] \ [Q]$
 $\langle \text{proof} \rangle$

end

— Evolution command (invariants)

lemma *R-g-ode-inv: diff-invariant* $I \ f \ T \ S \ t_0 \ G \implies [P] \leq [I] \implies [\lambda s. I \ s \wedge G \ s] \leq [Q] \implies$
 $(x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0 \ \text{DINV} \ I) \leq \text{Ref} \ [P] \ [Q]$
 $\langle \text{proof} \rangle$

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* $:: ((a::\text{banach}) \Rightarrow a) \Rightarrow 'a \ \text{pred} \Rightarrow 'a \ \text{nd-fun} \ ((\lambda x' = - \ \& \ -))$
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* $:: ((a::\text{banach}) \Rightarrow a) \Rightarrow 'a \ \text{pred} \Rightarrow 'a \ \text{pred} \Rightarrow 'a \ \text{nd-fun} \ ((\lambda x' = - \ \& \ - \ \text{DINV} \ -))$
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-rule1*:

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 $\{P\} \ (x' = f \ \& \ G) \ \{Q\}$

$\langle \text{proof} \rangle$

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 $\{P\} \ (x' = (\lambda s. c) \ \& \ G) \ \{Q\}$

$\langle \text{proof} \rangle$

lemma *diff-weak-rule*:

assumes $[G] \leq [Q]$

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

$\langle \text{proof} \rangle$

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 $\{P\} \ (x' = f \ \& \ G \ \text{on} \ U \ S \ @ \ t_0) \ \{Q\}$

$\langle \text{proof} \rangle$

lemma *diff-inv-rule*:

assumes $[P] \leq [I]$ **and** *diff-invariant* $I \ f \ U \ S \ t_0 \ G$ **and** $[I] \leq [Q]$

```

shows {P} (x' = f & G on U S @ t0) {Q}
<proof>

```

```

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

— Verified with annotated dynamics

lemma $pendulum-dyn: \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} EVOL \varphi G T \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\}$
 $\langle proof \rangle$

lemma $pendulum-inv: \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} (x'=f \ \& \ G) \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\}$
 $\langle proof \rangle$

lemma $local-flow-pend: local-flow f UNIV UNIV \varphi$
 $\langle proof \rangle$

lemma $pendulum-flow: \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\} (x'=f \ \& \ G) \{\lambda s. r^2 = (s \$ 1)^2 + (s \$ 2)^2\}$
 $\langle proof \rangle$

no-notation $fpend (f)$
and $pend-flow (\varphi)$

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 $ball-flow :: real \Rightarrow real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
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>

lemma *fball-invariant*:

fixes $g \ h :: real$
defines *divv*: $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows *diff-invariant I* $(\lambda t. f \ g) (\lambda s. UNIV) UNIV \ 0 \ G$
<proof>

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies$

$\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0) \ \text{DINV} \ (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2$
 $= 0));$
 $(IF \ (\lambda s. s\$1 = 0) \ \text{THEN} \ (2 ::= (\lambda s. - s\$2)) \ \text{ELSE} \ \text{skip}))$
 $INV \ (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \ \{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
<proof>

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$
shows $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
and $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
<proof>

lemma [*bb-real-arith*]:

assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
shows $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$
 $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs = ?rhs*)
<proof>

lemma *bouncing-ball-dyn*: $g < 0 \implies h \geq 0 \implies$

$\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
(LOOP
 $((EVOL \ (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$
 $(IF \ (\lambda s. s\$1 = 0) \ \text{THEN} \ (2 ::= (\lambda s. - s\$2)) \ \text{ELSE} \ \text{skip}))$
 $INV \ (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \ \{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
<proof>

lemma *local-flow-ball*: *local-flow* $(f \ g) UNIV UNIV (\varphi \ g)$

$\langle proof \rangle$

lemma *bouncing-ball-flow*: $g < 0 \implies h \geq 0 \implies$
 $\{\lambda s. s\$1 = h \wedge s\$2 = 0\}$
 $(LOOP$
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0));$
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \{\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h\}$
 $\langle proof \rangle$

lemma *R-bb-assign*: $g < (0::real) \implies 0 \leq h \implies$
 $2 ::= (\lambda s. - s\$2) \leq Ref$
 $[\lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
 $[\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$
 $\langle proof \rangle$

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows $Ref \ [\lambda s. s\$1 = h \wedge s\$2 = 0] \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h] \geq$
 $(LOOP$
 $((EVOL (\varphi \ g) (\lambda s. s\$1 \geq 0) T);$
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
 $\langle proof \rangle$

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 φ a L τ $s \equiv (\chi$ i . if $i = 1$ $then$ $-exp(-a * \tau) * (L - s\$1) + L$ $else$
 $(if$ $i = 2$ $then$ $\tau + s\$2$ $else$ $s\$i))$

— Verified with the flow

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

lemma *local-lipschitz-therm-dyn*:
assumes $0 < (a::real)$
shows *local-lipschitz UNIV UNIV* ($\lambda t::real. f\ a\ L$)
 $\langle proof \rangle$

lemma *local-flow-therm*: $a > 0 \implies local-flow\ (f\ a\ L)\ UNIV\ UNIV\ (\varphi\ a\ L)$
 $\langle proof \rangle$

lemma *therm-dyn-down-real-arith*:
assumes $a > 0$ **and** *Thyps*: $0 < Tmin$ $Tmin \leq T$ $T \leq Tmax$
and *thyps*: $0 \leq (\tau::real) \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(Tmin / T) / a)$
shows $Tmin \leq exp(-a * \tau) * T$ **and** $exp(-a * \tau) * T \leq Tmax$
 $\langle proof \rangle$

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

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}
 ⟨proof⟩

lemma *R-therm-dyn-down:*

assumes $a > 0$ and $0 \leq \tau$ and $0 < Tmin$ and $Tmax < L$
 shows $Ref \ [\lambda s. s\$4 = 0 \wedge I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$
 $(x' = (\lambda t. f a 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
 ⟨proof⟩

lemma *R-therm-dyn-up:*

assumes $a > 0$ and $0 \leq \tau$ and $0 < Tmin$ and $Tmax < L$
 shows $Ref \ [\lambda s. s\$4 \neq 0 \wedge I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$
 $(x' = (\lambda t. f a L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0)$
 ⟨proof⟩

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
 $(x' = (\lambda t. f a L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0))$
 ⟨proof⟩

lemma *R-therm-assign1:* $Ref \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \geq$
 $(2 ::= (\lambda s. 0))$
 ⟨proof⟩

lemma *R-therm-assign2:*

$Ref \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$
 $(3 ::= (\lambda s. s\$1))$
 ⟨proof⟩

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)
 ⟨*proof*⟩

lemma *R-therm-loop*: $\text{Ref } [I \ Tmin \ Tmax] [I \ Tmin \ Tmax] \geq$
 (*LOOP*
 $\text{Ref } [I \ Tmin \ Tmax] [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1]$;
 $\text{Ref } [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] [I \ Tmin \ Tmax]$
 $\text{INV } I \ Tmin \ Tmax$)
 ⟨*proof*⟩

lemma *R-thermostat-flow*:
 assumes $a > 0$ and $0 \leq \tau$ and $0 < Tmin$ and $Tmax < L$
 shows $\text{Ref } [I \ Tmin \ Tmax] [I \ Tmin \ Tmax] \geq$
 (*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*
 $(x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on} \ (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0)$
 ELSE
 $(x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on} \ (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0)$
) *INV* $I \ Tmin \ Tmax$)
)
 ⟨*proof*⟩

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 $hmin \leq h \leq hmax$. 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* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4 (\varphi)$
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* ($\varphi\ k$)
<proof>

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 $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$
<proof>

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
 $(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x' = (\lambda t.\ f\ (c_i - c_o)))\ \&\ G\ hmax\ (c_i - c_o)\ \text{on } (\lambda s.$
 $\{0.. \tau\})\ UNIV\ @\ 0)$
 $ELSE\ (x' = (\lambda t.\ f\ (-c_o))\ \&\ G\ hmin\ (-c_o)\ \text{on } (\lambda s.\ \{0.. \tau\})\ UNIV\ @\ 0))$
 $INV\ I\ hmin\ hmax)$
 $\{I\ hmin\ hmax\}$
<proof>

lemma *tank-diff-inv*:

$0 \leq \tau \implies \text{diff-invariant } (dI \text{ } hmin \text{ } hmax \text{ } k) (\lambda t. f \text{ } k) (\lambda s. \{0..\tau\}) \text{ UNIV } 0 \text{ Guard}$
 $\langle \text{proof} \rangle$

lemma *tank-inv-arith1*:

assumes $0 \leq (\tau::\text{real})$ **and** $c_o < c_i$ **and** $b: hmin \leq y_0$ **and** $g: \tau \leq (hmax - y_0) / (c_i - c_o)$
shows $hmin \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq hmax$
 $\langle \text{proof} \rangle$

lemma *tank-inv-arith2*:

assumes $0 \leq (\tau::\text{real})$ **and** $0 < c_o$ **and** $b: y_0 \leq hmax$ **and** $g: \tau \leq -((hmin - y_0) / c_o)$
shows $hmin \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq hmax$
 $\langle \text{proof} \rangle$

lemma *tank-inv*:

assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\{I \text{ } hmin \text{ } hmax\}$
 $(\text{LOOP}$
 — control
 $((2 ::= (\lambda s. 0)); (\beta ::= (\lambda s. s\$1)));$
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) \text{ THEN } (4 ::= (\lambda s. 1)) \text{ ELSE}$
 $(\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \text{ THEN } (4 ::= (\lambda s. 0)) \text{ ELSE skip}));$
 — dynamics
 $(\text{IF } (\lambda s. s\$4 = 0) \text{ THEN}$
 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ hmax (c_i - c_o) \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV } @ \ 0 \ \text{DINV}$
 $(dI \ hmin \ hmax (c_i - c_o)))$
 ELSE
 $(x' = (\lambda t. f (-c_o)) \ \& \ G \ hmin (-c_o) \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV } @ \ 0 \ \text{DINV } (dI$
 $hmin \ hmax (-c_o))))$
 $\text{INV } I \ hmin \ hmax$
 $\{I \ hmin \ hmax\}$
 $\langle \text{proof} \rangle$

abbreviation *tank-ctrl* $:: \text{real} \Rightarrow \text{real} \Rightarrow (\text{real}^4) \text{ nd-fun } (ctrl)$

where $ctrl \ hmin \ hmax \equiv ((2 ::= (\lambda s. 0)); (\beta ::= (\lambda s. s\$1)));$
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) \text{ THEN } (4 ::= (\lambda s. 1)) \text{ ELSE}$
 $(\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \text{ THEN } (4 ::= (\lambda s. 0)) \text{ ELSE skip}))$

abbreviation *tank-dyn-dinv* $:: \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow (\text{real}^4) \text{ nd-fun } (dyn)$

where $dyn \ c_i \ c_o \ hmin \ hmax \ \tau \equiv (\text{IF } (\lambda s. s\$4 = 0) \text{ THEN}$
 $(x' = (\lambda t. f (c_i - c_o)) \ \& \ G \ hmax (c_i - c_o) \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV } @ \ 0 \ \text{DINV}$
 $(dI \ hmin \ hmax (c_i - c_o)))$
 ELSE
 $(x' = (\lambda t. f (-c_o)) \ \& \ G \ hmin (-c_o) \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV } @ \ 0 \ \text{DINV } (dI$
 $hmin \ hmax (-c_o))))$

abbreviation *tank-dinv* $c_i \ c_o \ hmin \ hmax \ \tau \equiv \text{LOOP } (ctrl \ hmin \ hmax ; \text{dyn } c_i \ c_o$

$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

$(IF\ (\lambda s. s\$4 = 0)\ THEN$

$(x' = (\lambda t. f\ (c_i - c_o)) \ \&\ G\ hmax\ (c_i - c_o)\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0\ DINV$

$(dI\ hmin\ hmax\ (c_i - c_o)))$

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)

<proof>

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.