Verification Components for Hybrid Systems

Jonathan Julián Huerta y Munive

March 17, 2025

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 9
3.2 Differential Invariants 11
3.3 Picard-Lindeloef 13
3.4 Flows for ODEs 15

4 Verification components for hybrid systems **17**
4.1 Verification of regular programs 17
4.2 Verification of hybrid programs 20
4.3 Derivation of the rules of dL 22
4.4 Examples 23
4.4.1 Pendulum 24
4.4.2 Bouncing Ball 24

4.4.3 Thermostat . . . 26
4.4.4 Tank . . . 27

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 . . . 38

6 Verification components with MKA 40
6.1 Verification in AKA . . . 40
6.2 Relational model . . . 42
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 . . . 55
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 . . . 61
6.7.2 Bouncing Ball . . . 62
6.7.3 Thermostat . . . 64
6.7.4 Tank . . . 65

7 Verification components with KAT 66
7.1 Hoare logic in KAT . . . 67
7.2 refinement KAT . . . 69
7.3 Verification of hybrid programs . . . 70
7.3.1 Regular programs . . . 70
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 81
7.6.3 Thermostat 83
7.6.4 Water tank 86
7.6.5 Tank 86
7.7 Verification of hybrid programs 89
7.7.1 Regular programs 90
7.7.2 Evolution commands 93
7.8 Refinement Components 94
7.9 Derivation of the rules of dL 99
7.10 Examples 99
7.10.1 Pendulum 100
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-Lindeloef-Qualitative*
begin

— Notation

open-bundle *derivative-syntax*
begin
no-notation *has-vderiv-on* (**infix** ‹(*has'-vderiv'-on*)› *50*)
notation *has-derivative* (‹(*1*(*D* - ↦ (-))/ -)› [*65*,*65*] *61*)
and *has-vderiv-on* (‹(*1 D* - = (-)/ *on* -)› [*65*,*65*] *61*)
end

notation *norm* (‹‖-‖›)

2.1 Real numbers

lemma *abs-le-eq*:
shows (*r*::*real*) > *0* ⟹ (|*x*| < *r*) = (−*r* < *x* ∧ *x* < *r*)
and (*r*::*real*) > *0* ⟹ (|*x*| ≤ *r*) = (−*r* ≤ *x* ∧ *x* ≤ *r*)
⟨*proof*⟩

lemma *real-ivl-eqs*:
assumes *0* < *r*
shows *ball x r* = {*x*−*r*<−−< *x*+*r*} **and** {*x*−*r*<−−< *x*+*r*} = {*x*−*r*<..< *x*+*r*}
and *ball* (*r* / *2*) (*r* / *2*) = {*0*<−−<*r*} **and** {*0*<−−<*r*} = {*0*<..<*r*}
and *ball 0 r* = {−*r*<−−<*r*} **and** {−*r*<−−<*r*} = {−*r*<..<*r*}
and *cball x r* = {*x*−*r*−−*x*+*r*} **and** {*x*−*r*−−*x*+*r*} = {*x*−*r*..*x*+*r*}
and *cball* (*r* / *2*) (*r* / *2*) = {*0*−−*r*} **and** {*0*−−*r*} = {*0*..*r*}
and *cball 0 r* = {−*r*−−*r*} **and** {−*r*−−*r*} = {−*r*..*r*}
⟨*proof*⟩

lemma *is-interval-real-nonneg*[*simp*]: *is-interval* (*Collect* ((≤) (*0*::*real*)))
⟨*proof*⟩

lemma *norm-rotate-eq*[*simp*]:
fixes *x* :: *'a*:: {*banach*,*real-normed-field*}
shows (*x* ∗ *cos t* − *y* ∗ *sin t*)² + (*x* ∗ *sin t* + *y* ∗ *cos t*)² = *x*² + *y*²
and (*x* ∗ *cos t* + *y* ∗ *sin t*)² + (*y* ∗ *cos t* − *x* ∗ *sin t*)² = *x*² + *y*²

⟨*proof*⟩

lemma *sum-eq-Sum*:
assumes *inj-on f A*
shows $(\sum x{\in}A.\ f\ x) = (\sum\ \{f\ x\ |x.\ x \in A\})$
⟨*proof*⟩

lemma *triangle-norm-vec-le-sum*: $\|x\| \leq (\sum i{\in}UNIV.\ \|x\ \$\ i\|)$
⟨*proof*⟩

2.2 Single variable derivatives

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

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

lemma *vderiv-on-composeI*:
assumes $D\ f = f'\ on\ g\ `\ T$
and $D\ g = g'\ on\ T$
and $h = (\lambda t.\ g'\ t *_R f'\ (g\ t))$
shows $D\ (\lambda t.\ f\ (g\ t)) = h\ on\ T$
⟨*proof*⟩

lemma *vderiv-uminusI*[*poly-derivatives*]:
$D\ f = f'\ on\ T \Longrightarrow g = (\lambda t.\ -\ f'\ t) \Longrightarrow D\ (\lambda t.\ -\ f\ t) = g\ on\ T$
⟨*proof*⟩

lemma *vderiv-npowI*[*poly-derivatives*]:
fixes *f*::*real* ⇒ *real*
assumes $n \geq 1$ **and** $D\ f = f'\ on\ T$ **and** $g = (\lambda t.\ n * (f'\ t) * (f\ t)\hat{}(n-1))$
shows $D\ (\lambda t.\ (f\ t)\hat{}n) = g\ on\ T$
⟨*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 $h = (\lambda t.\ (f'\ t * g\ t - f\ t * (g'\ t))\ /\ (g\ t)\hat{}2)$
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 $a \neq 0 \Longrightarrow D\ (\lambda t.\ t/a) = (\lambda t.\ 1/a)\ on\ T$
⟨*proof*⟩

lemma $(a{::}real) \neq 0 \Longrightarrow D\ f = f'\ on\ T \Longrightarrow g = (\lambda t.\ (f'\ t)/a) \Longrightarrow D\ (\lambda t.\ (f\ t)/a) = g\ on\ T$
⟨*proof*⟩

lemma $\forall\, t{\in}T.\ f\ t \neq (0{::}real) \Longrightarrow D\ f = f'\ on\ T \Longrightarrow g = (\lambda t.\ -\ a * f'\ t\ /\ (f\ t)\hat{}2) \Longrightarrow$
$D\ (\lambda t.\ a/(f\ t)) = g\ on\ T$
⟨*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 $D\ x = x'\ on\ (\lambda \tau.\ \tau + t)\ `\ T \Longrightarrow D\ (\lambda \tau.\ x\ (\tau + t)) = (\lambda \tau.\ x'\ (\tau + t))\ on\ T$
⟨*proof*⟩

lemma $a \neq 0 \Longrightarrow D\ (\lambda t.\ t/a) = (\lambda t.\ 1/a)\ on\ T$
⟨*proof*⟩

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ a5 * t\hat{}5 + a3 * (t\hat{}3\ /\ c) - a2 * exp\ (t\hat{}2) + a1 * cos\ t + a0) =$
$(\lambda t.\ 5 * a5 * t\hat{}4 + 3 * a3 * (t\hat{}2\ /\ c) - 2 * a2 * t * exp\ (t\hat{}2) - a1 * sin\ t)\ on\ T$
⟨*proof*⟩

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ -\ a3 * exp\ (t\hat{}3\ /\ c) + a1 * sin\ t + a2 * t\hat{}2) =$
$(\lambda t.\ a1 * cos\ t + 2 * a2 * t - 3 * a3 * t\hat{}2\ /\ c * exp\ (t\hat{}3\ /\ c))\ on\ T$
⟨*proof*⟩

lemma $c \neq 0 \Longrightarrow D\ (\lambda t.\ exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c))) =$
$(\lambda t.\ -4 * a * t\hat{}3 * sin\ (t\hat{}4)\ /\ c * cos\ (cos\ (t\hat{}4)\ /\ c) * exp\ (a * sin\ (cos\ (t\hat{}4)\ /\ c)))\ on\ T$
⟨*proof*⟩

2.3 Intermediate Value Theorem

lemma *IVT-two-functions*:
fixes f :: ('a::{*linear-continuum-topology*, *real-vector*}) ⇒
('b::{*linorder-topology*,*real-normed-vector*,*ordered-ab-group-add*})
assumes *conts*: *continuous-on* {*a*..*b*} *f continuous-on* {*a*..*b*} *g*
and *ahyp*: $f\ a < g\ a$ **and** *bhyp*: $g\ b < f\ b$ **and** $a \leq b$
shows $\exists x{\in}\{a..b\}.\ f\ x = g\ x$
⟨*proof*⟩

lemma *IVT-two-functions-real-ivl*:
fixes f :: *real* ⇒ *real*
assumes *conts*: *continuous-on* {*a*−−*b*} *f continuous-on* {*a*−−*b*} *g*
and *ahyp*: $f\ a < g\ a$ **and** *bhyp*: $g\ b < f\ b$
shows $\exists x{\in}\{a--b\}.\ f\ x = g\ x$
⟨*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*::*real*) ∈ *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*::*finite*) ⇒ '*b* ⇒ *bool*
assumes *h*:∀ *i*. *eventually* (P *i*) F
shows *eventually* ($\lambda x.\ \forall i.\ P\ i\ x$) F
⟨*proof*⟩

lemma *eventually-all-finite-mono*:
fixes P :: ('*a*::*finite*) ⇒ '*b* ⇒ *bool*

assumes *h1*: $\forall i.$ *eventually* $(P\ i)\ F$
and *h2*: $\forall x.\ (\forall i.\ (P\ i\ x)) \longrightarrow Q\ x$
shows *eventually* $Q\ F$
⟨*proof*⟩

2.5 Multivariable derivatives

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

lemma *vec-upd-eq*: *vec-upd* $s\ i\ a = (\chi\ j.$ *if* $j = i$ *then* a *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*:
(*bounded-linear* f) $\longleftrightarrow$ $(\forall i.$ *bounded-linear* $(\lambda x.\ (f\ x)\ \$\ i))$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *open-preimage-nth*:
open $S \Longrightarrow$ *open* $\{s::('a::real\text{-}normed\text{-}vector\hat{}'n::finite).\ s\ \$\ i \in S\}$
⟨*proof*⟩

lemma *tendsto-nth-iff*: — following (*?f* $\longrightarrow$ *?l*) *?F* = $(\forall i{\in}Basis.\ ((\lambda x.\ ?f\ x \cdot i) \longrightarrow ?l \cdot i)\ ?F)$
fixes $l::'a::real\text{-}normed\text{-}vector\hat{}'n::finite$
defines $m \equiv real\ CARD('n)$
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* = $(\forall i{\in}Basis.\ D\ (\lambda x.\ ?f\ x \cdot i) \mapsto (\lambda x.\ ?f'\ x \cdot i)$ *at ?a within ?S*)
$(D\ f \mapsto f'$ *at* x *within* $S) \longleftrightarrow (\forall i.\ D\ (\lambda s.\ f\ s\ \$\ i) \mapsto (\lambda s.\ f'\ s\ \$\ i)$ *at* x *within* $S)$
⟨*proof*⟩

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

lemma *frechet-tendsto-vec-lambda*:
fixes $f::real \Rightarrow ('a::banach)\hat{}('m::finite)$ **and** $x::real$ **and** $T::real\ set$
defines $x_0 \equiv netlimit$ (*at* x *within* T) **and** $m \equiv real\ CARD('m)$
assumes $\forall i.\ ((\lambda y.\ (f\ y\ \$\ i - f\ x_0\ \$\ i - (y - x_0) *_R f'\ x\ \$\ i)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)$ (*at* x *within* T)
shows $((\lambda y.\ (f\ y - f\ x_0 - (y - x_0) *_R f'\ x)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)$ (*at* x *within* T)

⟨*proof*⟩

lemma *tendsto-norm-bound*:
∀ *x*. ‖*G x* − *L*‖ ≤ ‖*F x* − *L*‖ ⟹ (*F* ⟶ *L*) *net* ⟹ (*G* ⟶ *L*) *net*
⟨*proof*⟩

lemma *tendsto-zero-norm-bound*:
∀ *x*. ‖*G x*‖ ≤ ‖*F x*‖ ⟹ (*F* ⟶ *0*) *net* ⟹ (*G* ⟶ *0*) *net*
⟨*proof*⟩

lemma *frechet-tendsto-vec-nth*:
fixes *f*::*real* ⇒ ('*a*::*real-normed-vector*)^'*m*
assumes ((λ*x*. (*f x* − *f x*$_0$ − (*x* − *x*$_0$) ∗$_R$ *f*′ *t*) /$_R$ (‖*x* − *x*$_0$‖)) ⟶ *0*) (*at t within T*)
shows ((λ*x*. (*f x* $ *i* − *f x*$_0$ $ *i* − (*x* − *x*$_0$) ∗$_R$ *f*′ *t* $ *i*) /$_R$ (‖*x* − *x*$_0$‖)) ⟶ *0*) (*at t within T*)
⟨*proof*⟩

end

3 Ordinary Differential Equations

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

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

3.1 Initial value problems and orbits

notation *image* (‹*P*›)

lemma *image-le-pred*[*simp*]: (*P f A* ⊆ {*s*. *G s*}) = (∀ *x*∈*A*. *G* (*f x*))
⟨*proof*⟩

definition *ivp-sols* :: (*real* ⇒ '*a* ⇒ ('*a*::*real-normed-vector*)) ⇒ ('*a* ⇒ *real set*) ⇒ '*a set* ⇒
real ⇒ '*a* ⇒ (*real* ⇒ '*a*) *set* (‹*Sols*›)
where *Sols f U S t*$_0$ *s* = {*X* ∈ *U s* → *S*. (*D X* = (λ*t*. *f t* (*X t*)) *on U s*) ∧ *X t*$_0$ = *s* ∧ *t*$_0$ ∈ *U s*}

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

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

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

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

definition *g-orbit* :: $((\prime a{::}ord) \Rightarrow \prime b) \Rightarrow (\prime b \Rightarrow bool) \Rightarrow \prime a\ set \Rightarrow \prime b\ set\ (\langle\gamma\rangle)$
where $\gamma\ X\ G\ U = \bigcup\{\mathcal{P}\ X\ (down\ U\ t)\ |t.\ \mathcal{P}\ X\ (down\ U\ t) \subseteq \{s.\ G\ s\}\}$

lemma *g-orbit-eq*:
fixes $X{::}(\prime a{::}preorder) \Rightarrow \prime b$
shows $\gamma\ X\ G\ U = \{X\ t\ |t.\ t \in U \wedge (\forall \tau{\in}down\ U\ t.\ G\ (X\ \tau))\}$
⟨*proof*⟩

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

lemma *g-orbital-eq*: $g\text{-}orbital\ f\ G\ U\ S\ t_0\ s =$
$\{X\ t\ |t\ X.\ t \in U\ s \wedge \mathcal{P}\ X\ (down\ (U\ s)\ t) \subseteq \{s.\ G\ s\} \wedge X \in Sols\ f\ U\ S\ t_0\ s\ \}$
⟨*proof*⟩

lemma *g-orbitalI*:
assumes $X \in Sols\ f\ U\ S\ t_0\ s$
and $t \in U\ s$ **and** $(\mathcal{P}\ X\ (down\ (U\ s)\ t) \subseteq \{s.\ G\ s\})$
shows $X\ t \in g\text{-}orbital\ f\ G\ U\ S\ t_0\ s$
⟨*proof*⟩

lemma *g-orbitalD*:
assumes $s' \in g\text{-}orbital\ f\ G\ U\ S\ t_0\ s$
obtains X **and** t **where** $X \in Sols\ f\ U\ S\ t_0\ s$
and $X\ t = s'$ **and** $t \in U\ s$ **and** $(\mathcal{P}\ X\ (down\ (U\ s)\ t) \subseteq \{s.\ G\ s\})$
⟨*proof*⟩

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

$t_0\ s\}$
⟨*proof*⟩

lemma $X \in \mathit{Sols}\ f\ U\ S\ t_0\ s \Longrightarrow \gamma\ X\ G\ (U\ s) \subseteq \mathit{g\text{-}orbital}\ f\ G\ U\ S\ t_0\ s$
⟨*proof*⟩

lemma $\mathit{g\text{-}orbital}\ f\ G\ U\ S\ t_0\ s \subseteq \mathit{g\text{-}orbital}\ f\ (\lambda s.\ \mathit{True})\ U\ S\ t_0\ s$
⟨*proof*⟩

no-notation *g-orbit* (‹γ›)

3.2 Differential Invariants

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

lemma *diff-invariant-eq*: $\mathit{diff\text{-}invariant}\ I\ f\ U\ S\ t_0\ G =$
$(\forall s.\ I\ s \longrightarrow (\forall X{\in}\mathit{Sols}\ f\ U\ S\ t_0\ s.\ (\forall \tau{\in}(\mathit{down}\ (U\ s)\ t).\ G\ (X\ \tau)) \longrightarrow I\ (X\ t))))$
⟨*proof*⟩

lemma *diff-inv-eq-inv-set*:
$\mathit{diff\text{-}invariant}\ I\ f\ U\ S\ t_0\ G = (\forall s.\ I\ s \longrightarrow (\mathit{g\text{-}orbital}\ f\ G\ U\ S\ t_0\ s) \subseteq \{s.\ I\ s\})$
⟨*proof*⟩

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

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

lemma *diff-invariant-eq-rule* [*diff-invariant-rules*]:
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow \mathit{is\text{-}interval}\ (U\ s)$
and *dX*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ \mathit{on}\ U(X\ t_0)) \Longrightarrow (D\ (\lambda\tau.\ \mu(X\ \tau){-}\nu(X\ \tau)) = ((*_R)\ 0)\ \mathit{on}\ U(X\ t_0))$
shows $\mathit{diff\text{-}invariant}\ (\lambda s.\ \mu\ s = \nu\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

lemma *diff-invariant-leq-rule* [*diff-invariant-rules*]:
fixes μ::'*a*::*banach* ⇒ *real*
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow \mathit{is\text{-}interval}\ (U\ s)$
and *Gg*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ \mathit{on}\ U(X\ t_0)) \Longrightarrow (\forall \tau{\in}U(X\ t_0).\ \tau > t_0 \longrightarrow G\ (X\ \tau) \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau))$
and *Gl*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ \mathit{on}\ U(X\ t_0)) \Longrightarrow (\forall \tau{\in}U(X\ t_0).\ \tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))$
and *dX*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ \mathit{on}\ U(X\ t_0)) \Longrightarrow D\ (\lambda\tau.\ \mu(X\ \tau){-}\nu(X\ \tau)) = (\lambda\tau.\ \mu'(X\ \tau){-}\nu'(X\ \tau))\ \mathit{on}\ U(X\ t_0)$

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 $\mu::'a::banach \Rightarrow real$
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow$ *is-interval* $(U\ s)$
and *Gg*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall \tau \in U(X\ t_0).\ \tau > t_0 \longrightarrow G\ (X\ \tau) \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau))$
and *Gl*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow (\forall \tau \in U(X\ t_0).\ \tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))$
and *dX*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow D\ (\lambda\tau.\ \mu(X\ \tau) - \nu(X\ \tau)) = (\lambda\tau.\ \mu'(X\ \tau) - \nu'(X\ \tau))\ on\ U(X\ t_0)$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

lemma *diff-invariant-nleq-rule*:
fixes $\mu::'a::banach \Rightarrow real$
shows *diff-invariant* $(\lambda s.\ \neg\ \nu\ s \leq \mu\ s)\ f\ U\ S\ t_0\ G \longleftrightarrow$ *diff-invariant* $(\lambda s.\ \nu\ s > \mu\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

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

lemma *diff-invariant-neq-rule-converse*:
fixes $\mu::'a::banach \Rightarrow real$
assumes *Uhyp*: $\bigwedge s.\ s \in S \Longrightarrow$ *is-interval* $(U\ s)\ \bigwedge s\ t.\ s \in S \Longrightarrow t \in U\ s \Longrightarrow t_0 \leq t$
and *conts*: $\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow$ *continuous-on* $(\mathcal{P}\ X\ (U\ (X\ t_0)))\ \nu$
$\bigwedge X.\ (D\ X = (\lambda\tau.\ f\ \tau\ (X\ \tau))\ on\ U(X\ t_0)) \Longrightarrow$ *continuous-on* $(\mathcal{P}\ X\ (U\ (X\ t_0)))\ \mu$
and *dI*:*diff-invariant* $(\lambda s.\ \nu\ s \neq \mu\ s)\ f\ U\ S\ t_0\ G$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

lemma *diff-invariant-conj-rule* [*diff-invariant-rules*]:
assumes *diff-invariant* $I_1\ f\ U\ S\ t_0\ G$
and *diff-invariant* $I_2\ f\ U\ S\ t_0\ G$
shows *diff-invariant* $(\lambda s.\ I_1\ s \wedge I_2\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

lemma *diff-invariant-disj-rule* [*diff-invariant-rules*]:
assumes *diff-invariant* $I_1\ f\ U\ S\ t_0\ G$
and *diff-invariant* $I_2\ f\ U\ S\ t_0\ G$

shows *diff-invariant* $(\lambda s.\ I_1\ s \vee I_2\ s)\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

3.3 Picard-Lindeloef

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

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

sublocale *ll-on-open-it T f S* t_0
⟨*proof*⟩

lemma *ll-on-open*: *ll-on-open T f S*
⟨*proof*⟩

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

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

lemma *flow-has-vderiv-on-ex-ivl*:
assumes $s \in S$
shows *D flow* t_0 *s* = $(\lambda t.\ f\ t\ (\mathit{flow}\ t_0\ s\ t))$ *on ex-ivl s*
⟨*proof*⟩

lemma *flow-funcset-ex-ivl*:
assumes $s \in S$
shows *flow* t_0 *s* ∈ *ex-ivl s* → *S*
⟨*proof*⟩

lemma *flow-in-ivp-sols-ex-ivl*:
assumes $s \in S$
shows *flow* t_0 *s* ∈ *Sols f* $(\lambda s.$ *ex-ivl s*$)$ *S* t_0 *s*
⟨*proof*⟩

lemma *csols-eq*: *csols* t_0 *s* = $\{(x,\ t).\ t \in T \wedge\ x \in \mathit{Sols}\ f\ (\lambda s.\ \{t_0 -\!- t\})\ S\ t_0\ s\}$
⟨*proof*⟩

lemma *subset-ex-ivlI*:
$Y_1 \in \mathit{Sols}\ f\ (\lambda s.\ T)\ S\ t_0\ s \Longrightarrow \{t_0 -\!- t\} \subseteq T \Longrightarrow A \subseteq \{t_0 -\!- t\} \Longrightarrow A \subseteq$ *ex-ivl*

s
⟨*proof*⟩

lemma *unique-solution*: — proved for a subset of T for general applications
assumes $s \in S$ **and** $t_0 \in U$ **and** $t \in U$
and *is-interval U* **and** $U \subseteq$ *ex-ivl s*
and *xivp*: $D\ Y_1 = (\lambda t.\ f\ t\ (Y_1\ t))$ *on* $U\ Y_1\ t_0 = s\ Y_1 \in U \to S$
and *yivp*: $D\ Y_2 = (\lambda t.\ f\ t\ (Y_2\ t))$ *on* $U\ Y_2\ t_0 = s\ Y_2 \in U \to S$
shows $Y_1\ t = Y_2\ t$
⟨*proof*⟩

Applications of lemma *unique-solution*:

lemma *unique-solution-closed-ivl*:
assumes *xivp*: $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on* $\{t_0 -\!- t\}\ X\ t_0 = s\ X \in \{t_0 -\!- t\} \to S$ **and** $t \in T$
and *yivp*: $D\ Y = (\lambda t.\ f\ t\ (Y\ t))$ *on* $\{t_0 -\!- t\}\ Y\ t_0 = s\ Y \in \{t_0 -\!- t\} \to S$ **and** $s \in S$
shows $X\ t = Y\ t$
⟨*proof*⟩

lemma *solution-eq-flow*:
assumes *xivp*: $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on ex-ivl s* $X\ t_0 = s\ X \in$ *ex-ivl s* $\to S$
and $t \in$ *ex-ivl s* **and** $s \in S$
shows $X\ t =$ *flow* $t_0\ s\ t$
⟨*proof*⟩

lemma *ivp-unique-solution*:
assumes $s \in S$ **and** *ivl*: *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $t \in U\ s$
and *ivp1*: $Y_1 \in$ *Sols f U S* $t_0\ s$ **and** *ivp2*: $Y_2 \in$ *Sols f U S* $t_0\ s$
shows $Y_1\ t = Y_2\ t$
⟨*proof*⟩

lemma *g-orbital-orbit*:
assumes $s \in S$ **and** *ivl*: *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$
and *ivp*: $Y \in$ *Sols f U S* $t_0\ s$
shows *g-orbital f G U S* $t_0\ s =$ *g-orbit Y G* $(U\ s)$
⟨*proof*⟩

end

lemma *local-lipschitz-add*:
fixes *f1 f2* :: *real* ⇒ *'a*::*banach* ⇒ *'a*
assumes *local-lipschitz T S f1*
and *local-lipschitz T S f2*
shows *local-lipschitz T S* $(\lambda t\ s.\ f1\ t\ s + f2\ t\ s)$
⟨*proof*⟩

lemma *picard-lindeloef-add*: *picard-lindeloef f1 T S* $t_0 \Longrightarrow$ *picard-lindeloef f2 T S* $t_0 \Longrightarrow$

picard-lindeloef $(\lambda t\ s.\ f1\ t\ s + f2\ t\ s)\ T\ S\ t_0$
⟨*proof*⟩

lemma *picard-lindeloef-constant*: *picard-lindeloef* $(\lambda t\ s.\ c)$ *UNIV UNIV* t_0
⟨*proof*⟩

3.4 Flows for ODEs

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

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

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

lemma *eq-solution-ivl*:
assumes *xivp*: $D\ X = (\lambda t.\ f\ (X\ t))$ *on* $\{0--t\}\ X\ 0 = s\ X \in \{0--t\} \to S$
and *indom*: $t \in T\ s \in S$
shows $X\ t = \varphi\ t\ s$
⟨*proof*⟩

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 $(\lambda t.\ \varphi\ t\ s) \in T \to S$
⟨*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 $Sols\ (\lambda t.\ f)\ (\lambda s.\ T)\ S\ 0\ s = \{(\lambda t.\ \varphi\ t\ s)\}$
⟨*proof*⟩

lemma *additive-in-ivp-sols*:
assumes $s \in S$ **and** $\mathcal{P}\ (\lambda\tau.\ \tau + t)\ T \subseteq T$
shows $(\lambda\tau.\ \varphi\ (\tau + t)\ s) \in Sols\ (\lambda t.\ f)\ (\lambda s.\ T)\ S\ 0\ (\varphi\ (0 + t)\ s)$
⟨*proof*⟩

lemma *is-monoid-action*:
assumes $s \in S$ **and** $T = UNIV$
shows $\varphi\ 0\ s = s$ **and** $\varphi\ (t_1 + t_2)\ s = \varphi\ t_1\ (\varphi\ t_2\ s)$
⟨*proof*⟩

lemma *g-orbital-collapses*:
assumes $s \in S$ **and** *is-interval* $(U\ s)$ **and** $U\ s \subseteq T$ **and** $0 \in U\ s$

shows *g-orbital* ($\lambda t.\ f$) $G\ U\ S\ 0\ s = \{\varphi\ t\ s|\ t.\ t \in U\ s \wedge (\forall \tau \in down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s))\}$
⟨*proof*⟩

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

lemma *orbit-eq*:
assumes $s \in S$
shows $\gamma^{\varphi}\ s = \{\varphi\ t\ s|\ t.\ t \in T\}$
⟨*proof*⟩

lemma *true-g-orbit-eq*:
assumes $s \in S$
shows $g\text{-}orbit\ (\lambda t.\ \varphi\ t\ s)\ (\lambda s.\ True)\ T = \gamma^{\varphi}\ s$
⟨*proof*⟩

end

lemma *line-is-local-flow*:
$0 \in T \Longrightarrow is\text{-}interval\ T \Longrightarrow open\ T \Longrightarrow local\text{-}flow\ (\lambda\ s.\ c)\ T\ UNIV\ (\lambda\ t\ s.\ s + t *_R c)$
⟨*proof*⟩

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

unbundle *no rtrancl-syntax*

notation *Union* ($\langle \mu \rangle$)
and *g-orbital* ($\langle (1x´ =- \&\ -\ on\ -\ -\ @\ -) \rangle$)

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*: $P \leq Q \Longrightarrow |F]\ P \leq |F]\ Q$
⟨*proof*⟩

lemma *fbox-anti*: $\forall s.\ F_1\ s \subseteq F_2\ s \Longrightarrow |F_2]\ P \leq |F_1]\ P$
⟨*proof*⟩

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 $(\lambda s.\ I\ s \vee J\ s) \leq |F]\ (\lambda s.\ I\ s \vee J\ s)$
⟨*proof*⟩

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

lemma *fbox-eta*[*simp*]: *fbox skip* $P = P$
⟨*proof*⟩

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

lemma *fbox-test*[*simp*]: $(\lambda s.\ (\ |¿P?]\ Q)\ s) = (\lambda s.\ P\ s \longrightarrow Q\ s)$
⟨*proof*⟩

definition *vec-upd* :: $'a^{\frown}'n \Rightarrow 'n \Rightarrow 'a \Rightarrow 'a^{\frown}'n$
where *vec-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* $s\$j)$
⟨*proof*⟩

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

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

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

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

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

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

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*: $(f\ ;\ g)\ x = \bigcup\ \{g\ y\ |y.\ y \in f\ x\}$
⟨*proof*⟩

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

lemma *hoare-kcomp*:
assumes $P \leq |G]\ R$ $R \leq |F]\ Q$
shows $P \leq |G\ ;\ F]\ Q$
⟨*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 $\equiv (\lambda s.$ *if P s then X s else Y s*$)$

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

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 $P \leq |IF\ T\ THEN\ X\ ELSE\ Y]\ Q$
⟨*proof*⟩

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

lemma *kpower-base*:
shows *kpower f 0 s* $= \{s\}$ **and** *kpower f* $(Suc\ 0)\ s = f\ s$
⟨*proof*⟩

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

definition *kleene-star* :: $('a \Rightarrow 'a\ set) \Rightarrow ('a \Rightarrow 'a\ set)$ (‹(‹*notation*=‹*postfix kleene-star*››-*)› [*1000*] *999*)
where $(f^*)\ s = \bigcup\ \{kpower\ f\ n\ s\ |n.\ n \in UNIV\}$

lemma *kpower-inv*:
fixes $F :: 'a \Rightarrow 'a\ set$

assumes $\forall\, s.\ I\ s \longrightarrow (\forall\, s'.\ s' \in F\ s \longrightarrow I\ s')$
shows $\forall\, s.\ I\ s \longrightarrow (\forall\, s'.\ s' \in (kpower\ F\ n\ s) \longrightarrow I\ s')$
⟨*proof*⟩

lemma *kstar-inv*: $I \leq |F]\ I \Longrightarrow I \leq |F^*]\ I$
⟨*proof*⟩

lemma *fbox-kstarI*:
assumes $P \leq I$ **and** $I \leq Q$ **and** $I \leq |F]\ I$
shows $P \leq |F^*]\ Q$
⟨*proof*⟩

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

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

lemma *fbox-loopI*: $P \leq I \Longrightarrow I \leq Q \Longrightarrow I \leq |F]\ I \Longrightarrow P \leq |LOOP\ F\ INV\ I]\ Q$
⟨*proof*⟩

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

4.2 Verification of hybrid programs

Verification by providing evolution

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

lemma *fbox-g-evol*[*simp*]:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $|EVOL\ \varphi\ G\ U]\ Q = (\lambda s.\ (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

Verification by providing solutions

lemma *fbox-g-orbital*: $|x´=f\ \&\ G\ on\ U\ S\ @\ t_0]\ Q =$
$(\lambda s.\ \forall\, X{\in}Sols\ f\ U\ S\ t_0\ s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
⟨*proof*⟩

context *local-flow*
begin

lemma *fbox-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$

shows $|x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0]\ Q =$
$(\lambda\ s.\ s \in S \longrightarrow (\forall\, t{\in}(U\ s).\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

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

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

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

end

Verification with differential invariants

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

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

lemma *fbox-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq |x' = f\ \&\ G\ on\ U\ S\ @\ t_0]\ I$ **and** $I \leq Q$
shows $P \leq |x' = f\ \&\ G\ on\ U\ S\ @\ t_0]\ Q$
⟨*proof*⟩

lemma *fbox-diff-inv*[*simp*]:
$(I \leq |x' = f\ \&\ G\ on\ U\ S\ @\ t_0]\ I) = diff\text{-}invariant\ I\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

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

context *local-flow*
begin

lemma *fbox-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
shows *diff-invariant* $I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$

((λs. s ∈ S ⟶ I s) = |x´= (λt. f) & (λs. True) on U S @ 0] (λs. s ∈ S ⟶ I s))
⟨proof⟩

lemma *diff-inv-eq-inv-set*:
diff-invariant I (λt. f) (λs. T) S 0 (λs. True) = (∀ s. I s ⟶ γ^{φ} s ⊆ {s. I s})
⟨proof⟩

end

lemma *fbox-g-odei*: P ≤ I ⟹ I ≤ |x´= f & G on U S @ t_0] I ⟹ (λs. I s ∧ G s) ≤ Q ⟹
P ≤ |x´= f & G on U S @ t_0 DINV I] Q
⟨proof⟩

4.3 Derivation of the rules of dL

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

abbreviation *g-dl-ode* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a ⇒ 'a set
(‹(1x´=- & -)›) **where** (x´=f & G) ≡ (x´=(λt. f) & G on (λs. {t. t ≥ 0}) UNIV @ 0)

abbreviation *g-dl-ode-inv* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a ⇒ 'a set
(‹(1x´=- & - DINV -)›)
where (x´=f & G DINV I) ≡ (x´=(λt. f) & G on (λs. {t. t ≥ 0}) UNIV @ 0 DINV I)

lemma *diff-solve-axiom1*:
assumes *local-flow f UNIV UNIV* φ
shows |x´= f & G] Q =
(λs. ∀ t≥0. (∀ τ∈{0..t}. G (φ τ s)) ⟶ Q (φ t s))
⟨proof⟩

lemma *diff-solve-axiom2*:
fixes c::'a::{*heine-borel*, *banach*}
shows |x´=(λs. c) & G] Q =
(λs. ∀ t≥0. (∀ τ∈{0..t}. G (s + τ *$_R$ c)) ⟶ Q (s + t *$_R$ c))
⟨proof⟩

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and ∀ s. P s ⟶ (∀ t≥0. (∀ τ∈{0..t}. G (φ τ s)) ⟶ Q (φ t s))
shows P ≤ |x´= f & G] Q
⟨proof⟩

lemma *diff-weak-axiom1*: (|x´= f & G on U S @ t_0] G) s
⟨proof⟩

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

lemma *diff-weak-rule*: $G \leq Q \Longrightarrow P \leq |x' = f \ \&\ G \ on \ T \ S \ @ \ t_0] \ Q$
⟨*proof*⟩

lemma *fbox-g-orbital-eq-univD*:
assumes $|x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ C = (\lambda s. \ True)$
and $\forall \tau \in (down \ (U \ s) \ t). \ x \ \tau \in (x' = f \ \&\ G \ on \ U \ S \ @ \ t_0) \ s$
shows $\forall \tau \in (down \ (U \ s) \ t). \ C \ (x \ \tau)$
⟨*proof*⟩

lemma *diff-cut-axiom*:
assumes $|x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ C = (\lambda s. \ True)$
shows $|x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ Q = |x' = f \ \&\ (\lambda s. \ G \ s \wedge C \ s) \ on \ U \ S \ @ \ t_0] \ Q$
⟨*proof*⟩

lemma *diff-cut-rule*:
assumes *fbox-C*: $P \leq |x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ C$
and *fbox-Q*: $P \leq |x' = f \ \&\ (\lambda s. \ G \ s \wedge C \ s) \ on \ U \ S \ @ \ t_0] \ Q$
shows $P \leq |x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ Q$
⟨*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 $(\ |x' = f \ \&\ G] \ I) \ s$
⟨*proof*⟩

lemma *diff-inv-axiom2*:
assumes *picard-lindeloef* $(\lambda t. \ f) \ UNIV \ UNIV \ 0$
and $\bigwedge s. \ \{t::real. \ t \geq 0\} \subseteq picard\text{-}lindeloef.ex\text{-}ivl \ (\lambda t. \ f) \ UNIV \ UNIV \ 0 \ s$
and *diff-invariant* $I \ (\lambda t. \ f) \ (\lambda s. \ \{t::real. \ t \geq 0\}) \ UNIV \ 0 \ G$
shows $|x' = f \ \&\ G] \ I = |(\lambda s. \ \{x. \ s = x \wedge G \ s\})] \ I$
⟨*proof*⟩

lemma *diff-inv-rule*:
assumes $P \leq I$ **and** *diff-invariant* $I \ f \ U \ S \ t_0 \ G$ **and** $I \leq Q$
shows $P \leq |x' = f \ \&\ G \ on \ U \ S \ @ \ t_0] \ Q$
⟨*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* ⇒ *real^2* (‹*f*›)
  where $f\ s \equiv (\chi\ i.\ \textit{if}\ i = 1\ \textit{then}\ s\$2\ \textit{else}\ -s\$1)$

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

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

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real^2* ⇒ *real^2* (‹φ›)
where φ *g t s* ≡ (χ *i*. *if i* = *1 then g* ∗ *t* ^ *2*/*2* + *s*$*2* ∗ *t* + *s*$*1 else g* ∗ *t* + *s*$*2*)

— Verified with differential invariants.

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

lemma *inv-imp-pos-le*[*bb-real-arith*]:
assumes *0* > *g* **and** *inv*: *2* ∗ *g* ∗ *x* − *2* ∗ *g* ∗ *h* = *v* ∗ *v*
shows (*x*::*real*) ≤ *h*
⟨*proof*⟩

lemma *diff-invariant* (λ*s*. *2* ∗ *g* ∗ *s*$*1* − *2* ∗ *g* ∗ *h* − *s*$*2* ∗ *s*$*2* = *0*) (λ*t*. *f g*) (λ*s*. *UNIV*) *S* t_0 *G*
⟨*proof*⟩

lemma *bouncing-ball-inv*: *g* < *0* ⟹ *h* ≥ *0* ⟹
(λ*s*. *s*$*1* = *h* ∧ *s*$*2* = *0*) ≤
|*LOOP* (
(*x´*=(*f g*) & (λ *s*. *s*$*1* ≥ *0*) *DINV* (λ*s*. *2* ∗ *g* ∗ *s*$*1* − *2* ∗ *g* ∗ *h* − *s*$*2* ∗ *s*$*2* = *0*)) ;
(*IF* (λ *s*. *s*$*1* = *0*) *THEN* (*2* ::= (λ*s*. − *s*$*2*)) *ELSE skip*))
INV (λ*s*. *0* ≤ *s*$*1* ∧ *2* ∗ *g* ∗ *s*$*1* − *2* ∗ *g* ∗ *h* − *s*$*2* ∗ *s*$*2* = *0*)]
(λ*s*. *0* ≤ *s*$*1* ∧ *s*$*1* ≤ *h*)
⟨*proof*⟩

lemma *inv-conserv-at-ground*[*bb-real-arith*]:
assumes *invar*: *2* ∗ *g* ∗ *x* = *2* ∗ *g* ∗ *h* + *v* ∗ *v*
and *pos*: *g* ∗ τ^2 / *2* + *v* ∗ τ + (*x*::*real*) = *0*
shows *2* ∗ *g* ∗ *h* + (*g* ∗ τ + *v*) ∗ (*g* ∗ τ + *v*) = *0*
⟨*proof*⟩

lemma *inv-conserv-at-air*[*bb-real-arith*]:
assumes *invar*: *2* ∗ *g* ∗ *x* = *2* ∗ *g* ∗ *h* + *v* ∗ *v*
shows *2* ∗ *g* ∗ (*g* ∗ τ^2 / *2* + *v* ∗ τ + (*x*::*real*)) =
2 ∗ *g* ∗ *h* + (*g* ∗ τ + *v*) ∗ (*g* ∗ τ + *v*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *bouncing-ball-dyn*: *g* < *0* ⟹ *h* ≥ *0* ⟹
(λ*s*. *s*$*1* = *h* ∧ *s*$*2* = *0*) ≤
|*LOOP* (
(*EVOL* (φ *g*) (λ *s*. *s*$*1* ≥ *0*) *T*) ;
(*IF* (λ *s*. *s*$*1* = *0*) *THEN* (*2* ::= (λ*s*. − *s*$*2*)) *ELSE skip*))
INV (λ*s*. *0* ≤ *s*$*1* ∧*2* ∗ *g* ∗ *s*$*1* = *2* ∗ *g* ∗ *h* + *s*$*2* ∗ *s*$*2*)]
(λ*s*. *0* ≤ *s*$*1* ∧ *s*$*1* ≤ *h*)
⟨*proof*⟩

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

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
(λ*s*. *s*$*1* = *h* ∧ *s*$*2* = *0*) ≤
|*LOOP* (
(*x´*=(λ*t*. *f g*) & (λ *s*. *s*$*1* ≥ *0*) *on* (λ*s*. *UNIV*) *UNIV* @ *0*) ;
(*IF* (λ *s*. *s*$*1* = *0*) *THEN* (*2* ::= (λ*s*. − *s*$*2*)) *ELSE skip*))
INV (λ*s*. *0* ≤ *s*$*1* ∧*2* ∗ *g* ∗ *s*$*1* = *2* ∗ *g* ∗ *h* + *s*$*2* ∗ *s*$*2*)]
(λ*s*. *0* ≤ *s*$*1* ∧ *s*$*1* ≤ *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* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f a L s* ≡ (*χ i*. *if i* = *2 then 1 else* (*if i* = *1 then* − *a* ∗ (*s*$*1* − *L*) *else 0*))

abbreviation *temp-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*φ*›)
where *φ a L t s* ≡ (*χ i*. *if i* = *1 then* − *exp*(−*a* ∗ *t*) ∗ (*L* − *s*$*1*) + *L else*
(*if i* = *2 then t* + *s*$*2 else s*$*i*))

— Verified with the flow.

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

lemma *local-lipschitz-temp-dyn*:
assumes *0* < (*a*::*real*)
shows *local-lipschitz UNIV UNIV* (λ*t*::*real*. *f a L*)
⟨*proof*⟩

lemma *local-flow-temp*: *a* > *0* ⟹ *local-flow* (*f a L*) *UNIV UNIV* (*φ 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.fbox-g-ode-subset*[*OF local-flow-temp*]

lemma *thermostat*:
assumes $a > 0$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0) \leq$
$|LOOP$
— control
$((2 ::= (\lambda s.\ 0));(3 ::= (\lambda s.\ s\$1));$
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN\ (4 ::= (\lambda s.1))\ ELSE$
$(IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)\ THEN\ (4 ::= (\lambda s.0))\ ELSE\ skip));$
— dynamics
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN\ (x´= f\ a\ 0\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a))$
$ELSE\ (x´= f\ a\ L\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ ((L-Tmax)/(L-s\$3)))/a)))\ )$
$INV\ (\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1))]$
$(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax)$
⟨*proof*⟩

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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where $f\ k\ s \equiv (\chi\ i.\ if\ i = 2\ then\ 1\ else\ (if\ i = 1\ then\ k\ else\ 0))$

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

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

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

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

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

lemma *tank-arith*:
assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau \in \{0..\tau\}.\ \tau \leq -((hmin - y)\ /\ c_o) \Longrightarrow hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0..\tau\}.\ \tau \leq (hmax - y)\ /\ (c_i - c_o) \Longrightarrow (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) * \tau + y$
and $y \leq hmax \Longrightarrow y - c_o * \tau \leq hmax$
⟨*proof*⟩

lemma *tank-flow*:
assumes $0 < c_o$ **and** $c_o < c_i$
shows $I\ hmin\ hmax \leq$
|*LOOP*
— control
(2 ::=(λs.0));(3 ::=(λs. s$1));
(*IF* (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) *THEN* (4 ::= (λs.1)) *ELSE*
(*IF* (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) *THEN* (4 ::= (λs.0)) *ELSE skip*));
— dynamics
(*IF* (λs. s$4 = 0) *THEN* (x´= f ($c_i$−$c_o$) & (G hmax ($c_i$−$c_o$)))
ELSE (x´= f (−c_o) & (G hmin (−c_o))))) *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 $fb_{\mathcal{F}}\ F\ S = (\bigcap \circ \mathcal{P}\ (-\ op_K\ F))\ (-\ S)$
⟨*proof*⟩

lemma $fb_{\mathcal{F}}\ F\ S = \{s.\ F\ s \subseteq S\}$
⟨*proof*⟩

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

lemma *ffb-iso*: $P \leq Q \Longrightarrow fb_{\mathcal{F}}\ F\ P \leq fb_{\mathcal{F}}\ F\ Q$
⟨*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 pred* ⇒ *'a* ⇒ *'a set* (‹(*1¿-?*)›)
where *¿P?* = (λ*s*. {*x*. *x* = *s* ∧ *P x*})

lemma *ffb-test*[*simp*]: *fb*$_{\mathcal{F}}$ *¿P?* *Q* = {*s*. *P s* ⟶ *s* ∈ *Q*}
⟨*proof*⟩

definition *assign* :: *'n* ⇒ (*'a*^*'n* ⇒ *'a*) ⇒ (*'a*^*'n*) ⇒ (*'a*^*'n*) *set* (‹(*2*- ::= -)› [*70*, *65*] *61*)
where (*x* ::= *e*) = (λ*s*. {*vec-upd s x* (*e s*)})

lemma *ffb-assign*[*simp*]: *fb*$_{\mathcal{F}}$ (*x* ::= *e*) *Q* = {*s*. (χ *j*. (((*$*) *s*)(*x* := (*e s*))) *j*) ∈ *Q*}
⟨*proof*⟩

definition *nondet-assign* :: *'n* ⇒ *'a*^*'n* ⇒ (*'a*^*'n*) *set* (‹(*2*- ::= *?*)› [*70*] *61*)
where (*x* ::= *?*) = (λ*s*. {(*vec-upd s x k*)|*k*. *True*})

lemma *fbox-nondet-assign*[*simp*]: *fb*$_{\mathcal{F}}$ (*x* ::= *?*) *P* = {*s*. ∀ *k*. (χ *j*. *if j* = *x then k else s*$*j*) ∈ *P*}
⟨*proof*⟩

lemma *ffb-choice*: *fb*$_{\mathcal{F}}$ (λ*s*. *F s* ∪ *G s*) *P* = *fb*$_{\mathcal{F}}$ *F P* ∩ *fb*$_{\mathcal{F}}$ *G P*
⟨*proof*⟩

lemma *le-ffb-choice-iff*: *P* ⊆ *fb*$_{\mathcal{F}}$ (λ*s*. *F s* ∪ *G s*) *Q* ⟷ *P* ⊆ *fb*$_{\mathcal{F}}$ *F Q* ∧ *P* ⊆ *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* ≤ *fb*$_{\mathcal{F}}$ *F R R* ≤ *fb*$_{\mathcal{F}}$ *G Q*
shows *P* ≤ *fb*$_{\mathcal{F}}$ (*F* ; *G*) *Q*
⟨*proof*⟩

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

lemma *ffb-if-then-else*[*simp*]:
fb$_{\mathcal{F}}$ (*IF T THEN X ELSE Y*) *Q* = {*s*. *T s* ⟶ *s* ∈ *fb*$_{\mathcal{F}}$ *X Q*} ∩ {*s*. ¬ *T s* ⟶ *s* ∈ *fb*$_{\mathcal{F}}$ *Y Q*}
⟨*proof*⟩

lemma *hoare-if-then-else*:
assumes *P* ∩ {*s*. *T s*} ≤ *fb*$_{\mathcal{F}}$ *X Q*
and *P* ∩ {*s*. ¬ *T s*} ≤ *fb*$_{\mathcal{F}}$ *Y Q*

shows $P \leq fb_{\mathcal{F}}\ (IF\ T\ THEN\ X\ ELSE\ Y)\ Q$
⟨*proof*⟩

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

lemma *kstar-inv*: $I \leq fb_{\mathcal{F}}\ F\ I \Longrightarrow I \subseteq fb_{\mathcal{F}}\ (kstar\ F)\ I$
⟨*proof*⟩

lemma *ffb-kstarI*:
assumes $P \leq I$ **and** $I \leq Q$ **and** $I \leq fb_{\mathcal{F}}\ F\ I$
shows $P \leq fb_{\mathcal{F}}\ (kstar\ F)\ Q$
⟨*proof*⟩

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*: $P \leq \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq Q \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ I\ s\} \Longrightarrow P \leq fb_{\mathcal{F}}\ (LOOP\ F\ INV\ I)\ Q$
⟨*proof*⟩

lemma *ffb-loopI-break*:
$P \leq fb_{\mathcal{F}}\ Y\ \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ X\ \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (Y\ ;\ (LOOP\ X\ INV\ I))\ Q$
⟨*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 $EVOL\ \varphi\ G\ U = (\lambda s.\ g\text{-}orbit\ (\lambda t.\ \varphi\ t\ s)\ G\ (U\ s))$

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

Verification by providing solutions

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

context *local-flow*
begin

lemma *ffb-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows $fb_{\mathcal{F}}\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ Q =$
$\{s.\ s \in S \longrightarrow (\forall\, t{\in}(U\ s).\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$
⟨*proof*⟩

lemma *ffb-g-ode*: $fb_{\mathcal{F}}\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ Q =$
$\{s.\ s \in S \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$ (**is** - = *?wlp*)
⟨*proof*⟩

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

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

end

Verification with differential invariants

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

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

lemma *ffb-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ I$ **and** $I \leq Q$
shows $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ Q$
⟨*proof*⟩

lemma *ffb-diff-inv*[*simp*]:
$(\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ I\ s\}) =$ *diff-invariant* $I\ f\ U\ S\ t_0\ G$
⟨*proof*⟩

lemma *bdf-diff-inv*:
diff-invariant $I\ f\ U\ S\ t_0\ G = (bd_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ I\ s\} \leq \{s.\ I\ s\})$
⟨*proof*⟩

lemma *diff-inv-guard-ignore*:

assumes $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ t_0)\ \{s.\ I\ s\}$
shows $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ I\ s\}$
⟨*proof*⟩

context *local-flow*
begin

lemma *ffb-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows *diff-invariant* $I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$
$(\{s.\ s \in S \longrightarrow I\ s\} = fb_{\mathcal{F}}\ (x' = (\lambda t.\ f)\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ 0)\ \{s.\ s \in S \longrightarrow I\ s\})$
⟨*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 *ffb-g-odei*: $P \leq \{s.\ I\ s\} \Longrightarrow \{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ I\ s\} \Longrightarrow$
$\{s.\ I\ s \wedge G\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I)\ Q$
⟨*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::*banach*)⇒'a) ⇒ 'a *pred* ⇒ 'a ⇒ 'a *set* (‹(*1x'=- & -*)›)
where $(x' = f\ \&\ G) \equiv (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

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

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

lemma *diff-solve-axiom2*:
fixes *c*::'a::{*heine-borel*, *banach*}
shows $fb_{\mathcal{F}}\ (x' = (\lambda s.\ c)\ \&\ G)\ Q =$
$\{s.\ \forall t \geq 0.\ (\forall \tau \in \{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow (s + t *_R c) \in Q\}$

⟨*proof*⟩

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
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 $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G)\ Q$
⟨*proof*⟩

lemma *diff-weak-axiom1*: $s \in (fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ G\ s\})$
⟨*proof*⟩

lemma *diff-weak-axiom2*: $fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q = fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ G\ s \longrightarrow s \in Q\}$
⟨*proof*⟩

lemma *diff-weak-rule*: $\{s.\ G\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q$
⟨*proof*⟩

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

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

lemma *diff-cut-rule*:
assumes *ffb-C*: $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{s.\ C\ s\}$
and *ffb-Q*: $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ Q$
shows $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ Q$
⟨*proof*⟩

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-lindeloef* $(\lambda t.\ f)\ UNIV\ UNIV\ 0$
and $\bigwedge s.\ \{t{::}real.\ t \geq 0\} \subseteq picard\text{-}lindeloef.ex\text{-}ivl\ (\lambda t.\ f)\ UNIV\ UNIV\ 0\ s$
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 $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ Q$
⟨*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 $x'\ t = y\ t$ and text "y' t = - x t" describe the circular motion of a mass attached to a string looked from above. We use *s$1* to represent the x-coordinate and *s$2* for the y-coordinate. We prove that this motion remains circular.

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

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

— Verified by providing the dynamics

lemma *pendulum-dyn*: $\{s.\ r^2 = (s\$1)^2 + (s\$2)^2\} \leq fb_{\mathcal{F}}\ (EVOL\ \varphi\ G\ T)\ \{s.\ r^2 = (s\$1)^2 + (s\$2)^2\}$
⟨*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* ⇒ *real^2* ⇒ *real^2* (‹*f*›)
  where *f g s* ≡ (*χ i. if i = 1 then s$2 else g*)

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real^2* ⇒ *real^2* (‹*φ*›)
  where *φ g t s* ≡ (*χ i. if i = 1 then g * t ^ 2/2 + s$2 * t + s$1 else g * t + s$2*)

— Verified with differential invariants.

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

lemma *inv-imp-pos-le*[*bb-real-arith*]:
  assumes *0 > g* **and** *inv*: *2 * g * x − 2 * g * h = v * v*
  shows (*x*::*real*) ≤ *h*
⟨*proof*⟩

lemma *bouncing-ball-inv*: *g < 0* ⟹ *h ≥ 0* ⟹
  {*s. s$1 = h ∧ s$2 = 0*} ≤ *fb*$_{\mathcal{F}}$
  (*LOOP* (
    (*x´=(f g) & (λ s. s$1 ≥ 0) DINV (λs. 2 * g * s$1 − 2 * g * h − s$2 * s$2 = 0)*) ;
    (*IF* (*λ s. s$1 = 0*) *THEN* (*2 ::= (λs. − s$2)*) *ELSE skip*))
  INV (*λs. 0 ≤ s$1 ∧2 * g * s$1 − 2 * g * h − s$2 * s$2 = 0*))
  {*s. 0 ≤ s$1 ∧ s$1 ≤ h*}
  ⟨*proof*⟩

lemma *inv-conserv-at-ground*[*bb-real-arith*]:
  assumes *invar*: *2 * g * x = 2 * g * h + v * v*
    and *pos*: *g * τ^2 / 2 + v * τ + (x::real) = 0*
  shows *2 * g * h + (g * τ * (g * τ + v) + v * (g * τ + v)) = 0*
⟨*proof*⟩

lemma *inv-conserv-at-air*[*bb-real-arith*]:
  assumes *invar*: *2 * g * x = 2 * g * h + v * v*
  shows *2 * g * (g * τ^2 / 2 + v * τ + (x::real)) =*
  *2 * g * h + (g * τ + v) * (g * τ + v)* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *bouncing-ball-dyn*: *g < 0* ⟹ *h ≥ 0* ⟹

$\{s.\ s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$
$(LOOP\ (EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T)\ ;\ (IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\{s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
⟨*proof*⟩

lemma *local-flow-ball*: $local\text{-}flow\ (f\ g)\ UNIV\ UNIV\ (\varphi\ g)$
⟨*proof*⟩

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{s.\ s\$1 = h \wedge s\$2 = 0\} \leq fb_{\mathcal{F}}$
$(LOOP\ (x'=(\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0)\ ;\ (IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\{s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
⟨*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 *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -\ a * (T - U)$ where U is $L \geq 0$ when the heater is on, and *0* when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, *3* is the temperature detected by the thermometer, and *4* states whether the heater is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

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

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

— Verified with the flow.

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

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*: $a > 0 \Longrightarrow$ *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 *ffb-temp-dyn* = *local-flow.ffb-g-ode-ivl*[*OF local-flow-temp - UNIV-I*]

lemma *thermostat*:
assumes $a > 0$ **and** $0 \leq t$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $\{s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0\} \leq fb_{\mathcal{F}}$
(*LOOP*
— control
((*2* ::= $(\lambda s.\ 0)$);(*3* ::= $(\lambda s.\ s\$1)$);
(*IF* $(\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)$ *THEN* (*4* ::= $(\lambda s.1)$) *ELSE*
(*IF* $(\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)$ *THEN* (*4* ::= $(\lambda s.0)$) *ELSE skip*));
— dynamics
(*IF* $(\lambda s.\ s\$4 = 0)$ *THEN* $(x' = (\lambda t.\ f\ a\ 0)\ \&\ (\lambda s.\ s\$2 \leq -\ (ln\ (Tmin/s\$3))/a)$
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 $(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
$\{s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax\}$
⟨*proof*⟩

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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f k s* ≡ (χ *i*. *if i = 2 then 1 else* (*if i = 1 then k else 0*))

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

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

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

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

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

lemma *tank-arith*:
assumes *0* ≤ (*τ*::*real*) **and** *0* < c_o **and** $c_o < c_i$
shows ∀*τ*∈{*0*..*τ*}. *τ* ≤ − ((*hmin* − *y*) / c_o) ⟹ *hmin* ≤ *y* − c_o ∗ *τ*
and ∀*τ*∈{*0*..*τ*}. *τ* ≤ (*hmax* − *y*) / (c_i − c_o) ⟹ (c_i − c_o) ∗ *τ* + *y* ≤ *hmax*
and *hmin* ≤ *y* ⟹ *hmin* ≤ (c_i − c_o) ∗ *τ* + *y*
and *y* ≤ *hmax* ⟹ *y* − c_o ∗ *τ* ≤ *hmax*
⟨*proof*⟩

lemma *tank-flow*:
assumes *0* < c_o **and** $c_o < c_i$
shows *Collect* (*I hmin hmax*) ≤ $fb_{\mathcal{F}}$
(*LOOP*
— control
((*2* ::=(λ*s*.*0*));(*3* ::=(λ*s*. *s$1*));
(*IF* (λ*s*. *s$4 = 0* ∧ *s$3* ≤ *hmin* + *1*) *THEN* (*4* ::= (λ*s*.*1*)) *ELSE*
(*IF* (λ*s*. *s$4 = 1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s*. *s$4 = 0*) *THEN* (*x´*= *f* (c_i−c_o) & (*G hmax* (c_i−c_o)))
ELSE (*x´*= *f* (−c_o) & (*G hmin* (−c_o))))) *INV I hmin hmax*)
(*Collect* (*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

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* (‹(‹*notation*=‹*mixfix if expression*››*if* (-)/ *then* (-)/ *else* (-))› [*0*, *0*, *10*] *10*)

notation *zero-class.zero* (‹*0*›)

context *antidomain-kleene-algebra*
begin

— Skip

lemma $|1]\ x = d\ x$
  ⟨*proof*⟩

lemma $|0]\ q = 1$
  ⟨*proof*⟩

lemma $|x \cdot y]\ q = |x]\ |y]\ q$
  ⟨*proof*⟩

declare *fbox-mult* [*simp*]

— Nondeterministic choice

lemma $|x + y]\ q = |x]\ q \cdot |y]\ q$

⟨*proof*⟩

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

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*: $ad\ p + |x]\ q = |d\ p \cdot x]\ q$
⟨*proof*⟩

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

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

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

lemma *fbox-frame*: $d\ p \cdot x \leq x \cdot d\ p \Longrightarrow d\ q \leq |x]\ r \Longrightarrow d\ p \cdot d\ q \leq |x]\ (d\ p \cdot d\ r)$
⟨*proof*⟩

lemma *fbox-shunt*: $d\ p \cdot d\ q \leq |x]\ t \longleftrightarrow d\ p \leq ad\ q + |x]\ t$
⟨*proof*⟩

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

lemma *fbox-while*: $d\ p \cdot d\ t \leq |x]\ p \Longrightarrow d\ p \leq |(d\ t \cdot x)^\star \cdot ad\ t]\ (d\ p \cdot ad\ t)$
⟨*proof*⟩

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*: $p \leq |x]\ p' \Longrightarrow p' \leq |y]\ q \Longrightarrow\ p \leq |x \cdot y]\ q$
⟨*proof*⟩

lemma *fbox-whilei-break*:
$d\ p \leq |y]\ i \Longrightarrow d\ i \cdot ad\ t \leq d\ q \Longrightarrow d\ i \cdot d\ t \leq |x]\ i \Longrightarrow d\ p \leq |y \cdot (while\ t\ do\ x\ inv\ i)]\ q$
⟨*proof*⟩

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

where *loop x inv i* = $x^\star$

lemma $d\ p \leq |x]\ p \Longrightarrow d\ p \leq |x^\star]\ p$
⟨*proof*⟩

lemma *fbox-loopi*: $p \leq d\ i \Longrightarrow d\ i \leq |x]\ i \Longrightarrow d\ i \leq d\ q \Longrightarrow p \leq |loop\ x\ inv\ i]\ q$
⟨*proof*⟩

lemma *fbox-loopi-break*:
$p \leq |y]\ d\ i \Longrightarrow d\ i \leq |x]\ i \Longrightarrow d\ i \leq d\ q \Longrightarrow p \leq |y \cdot (loop\ x\ inv\ i)]\ q$
⟨*proof*⟩

lemma $p \leq i \Longrightarrow i \leq |x]i \Longrightarrow i \leq q \Longrightarrow p \leq |x]q$
⟨*proof*⟩

lemma $p \leq d\ i \Longrightarrow d\ i \leq |x]i \Longrightarrow i \leq d\ q \Longrightarrow p \leq |x]q$
⟨*proof*⟩

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

⟨*proof*⟩

lemma $d\ i \leq |x]\ i \Longrightarrow d\ j \leq |x]\ j \Longrightarrow (d\ i + d\ j) \leq |x]\ (d\ i + d\ j)$
⟨*proof*⟩

lemma *plus-inv*: $i \leq |x]\ i \Longrightarrow j \leq |x]\ j \Longrightarrow (i + j) \leq |x]\ (i + j)$
⟨*proof*⟩

lemma *mult-inv*: $d\ i \leq |x]\ i \Longrightarrow d\ j \leq |x]\ j \Longrightarrow (d\ i \cdot d\ j) \leq |x]\ (d\ i \cdot d\ j)$
⟨*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 *dioid-one-zero*
begin

lemma *power-inductl*: $z + x \cdot y \leq y \Longrightarrow (x\ \widehat{}\ n) \cdot z \leq y$
⟨*proof*⟩

lemma *power-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot (x \hat{} n) \leq y$
⟨*proof*⟩

end

interpretation *rel-dioid*: *dioid-one-zero* $(\cup)$ (O) Id $\{\}$ $(\subseteq)$ $(\subset)$
⟨*proof*⟩

lemma *power-is-relpow*: *rel-dioid.power* X $n = X \hat{\hat{}} n$
⟨*proof*⟩

lemma *rel-star-def*: $X\hat{}* = (\bigcup n.$ *rel-dioid.power* X $n)$
⟨*proof*⟩

lemma *rel-star-contl*: X O $Y\hat{}* = (\bigcup n.$ X O *rel-dioid.power* Y $n)$
⟨*proof*⟩

lemma *rel-star-contr*: $X\hat{}*$ O $Y = (\bigcup n.$ (*rel-dioid.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* ⇒ *'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 ⇒ bool*

unbundle *no floor-ceiling-syntax*
no-notation *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 ⇒ 'a rel* (‹(*1*⌈-⌉)›) **where**
  ⌈*P*⌉ = {(*s*,*s*) |*s*. *P s*}

lemma *p2r-simps*[*simp*]:
  ⌈*P*⌉ ≤ ⌈*Q*⌉ = (∀ *s*. *P s* ⟶ *Q s*)
  (⌈*P*⌉ = ⌈*Q*⌉) = (∀ *s*. *P s* = *Q s*)
  (⌈*P*⌉ ; ⌈*Q*⌉) = ⌈λ *s*. *P s* ∧ *Q s*⌉
  (⌈*P*⌉ ∪ ⌈*Q*⌉) = ⌈λ *s*. *P s* ∨ *Q s*⌉
  rel-ad ⌈*P*⌉ = ⌈λ*s*. ¬ *P s*⌉
  rel-aka.ads-d ⌈*P*⌉ = ⌈*P*⌉
  ⟨*proof*⟩

lemma *in-p2r* [*simp*]: (*a*,*b*) ∈ ⌈*P*⌉ = (*P a* ∧ *a* = *b*)
  ⟨*proof*⟩

Lemmas for verification condition generation

lemma *wp-rel*: *wp R* ⌈*P*⌉ = ⌈λ *x*. ∀ *y*. (*x*,*y*) ∈ *R* ⟶ *P y*⌉
  ⟨*proof*⟩

lemma *wp-test*[*simp*]: *wp* ⌈*P*⌉ ⌈*Q*⌉ = ⌈λ*s*. *P s* ⟶ *Q s*⌉
  ⟨*proof*⟩

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

lemma *wp-assign* [*simp*]: *wp* (*x* ::= *e*) ⌈*Q*⌉ = ⌈λ*s*. *Q* (χ *j*. ((($) *s*)(*x* := (*e s*))) *j*)⌉
  ⟨*proof*⟩

definition *nondet-assign* :: *'b ⇒* (*'a^'b*) *rel* (‹(*2*- ::= *?*)› [*70*] *61*)
  where (*x* ::= *?*) = {(*s*,*vec-upd s x k*)|*s k*. *True*}

lemma *wp-nondet-assign*[*simp*]: *wp* (*x* ::= *?*) ⌈*P*⌉ = ⌈λ*s*. ∀ *k*. *P* (χ *j*. ((($) *s*)(*x* := *k*)) *j*)⌉
  ⟨*proof*⟩

lemma *le-wp-choice-iff*: ⌈*P*⌉ ≤ *wp* (*X* ∪ *Y*) ⌈*Q*⌉ ⟷ ⌈*P*⌉ ≤ *wp X* ⌈*Q*⌉ ∧ ⌈*P*⌉ ≤

wp Y ⌈Q⌉
⟨*proof*⟩

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

— Finite iteration

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

lemma *change-loopI*: *LOOP X INV G = LOOP X INV I*
⟨*proof*⟩

lemma *wp-loopI*:
⌈P⌉ ≤ ⌈I⌉ ⟹ ⌈I⌉ ≤ ⌈Q⌉ ⟹ ⌈I⌉ ≤ wp R ⌈I⌉ ⟹ ⌈P⌉ ≤ wp (LOOP R INV I) ⌈Q⌉
⟨*proof*⟩

lemma *wp-loopI-break*:
⌈P⌉ ≤ wp Y ⌈I⌉ ⟹ ⌈I⌉ ≤ wp X ⌈I⌉ ⟹ ⌈I⌉ ≤ ⌈Q⌉ ⟹ ⌈P⌉ ≤ wp (Y ; (LOOP X INV I)) ⌈Q⌉
⟨*proof*⟩

6.3.2 Evolution commands

Verification by providing evolution

definition *g-evol* :: *((′a::ord) ⇒ ′b ⇒ ′b) ⇒ ′b pred ⇒ (′b ⇒ ′a set) ⇒ ′b rel* (‹*EVOL*›)
where *EVOL φ G U = {(s,s′) |s s′. s′ ∈ g-orbit (λt. φ t s) G (U s)}*

lemma *wp-g-dyn*[*simp*]:
fixes *φ* :: *(′a::preorder) ⇒ ′b ⇒ ′b*
shows *wp (EVOL φ G U) ⌈Q⌉ = ⌈λs. ∀ t∈U s. (∀ τ∈down (U s) t. G (φ τ s)) ⟶ Q (φ t s)⌉*
⟨*proof*⟩

Verification by providing solutions

definition *g-ode* :: *(real ⇒ (′a::banach)⇒′a) ⇒ ′a pred ⇒ (′a ⇒ real set) ⇒ ′a set ⇒ real ⇒*
′a rel (‹*(1x´=- & - on - - @ -)*›)
where *(x´= f & G on U S @ t₀) = {(s,s′) |s s′. s′ ∈ g-orbital f G U S t₀ s}*

lemma *wp-g-orbital*: *wp (x´= f & G on U S @ t₀) ⌈Q⌉ =*
⌈λs. ∀ X∈Sols f U S t₀ s. ∀ t∈U s. (∀ τ∈down (U s) t. G (X τ)) ⟶ Q (X t)⌉
⟨*proof*⟩

context *local-flow*
begin

lemma *wp-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows *wp* $(x' = (\lambda t.\ f)\ \&\ G$ *on* $U\ S\ @\ 0)\ \lceil Q \rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall\, t \in (U\ s).\ (\forall\, \tau \in$ *down* $(U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
⟨*proof*⟩

lemma *wp-g-ode*: *wp* $(x' = (\lambda t.\ f)\ \&\ G$ *on* $(\lambda s.\ T)\ S\ @\ 0)\ \lceil Q \rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall\, t \in T.\ (\forall\, \tau \in$ *down* $T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
⟨*proof*⟩

lemma *wp-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow$ *wp* $(x' = (\lambda t.\ f)\ \&\ G$ *on* $(\lambda s.\ \{0..t\})$ $S\ @\ 0)\ \lceil Q \rceil =$
$\lceil \lambda s.\ s \in S \longrightarrow (\forall\, t \in \{0..t\}.\ (\forall\, \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
⟨*proof*⟩

lemma *wp-orbit*: *wp* $(\{(s,s') \mid s\ s'.\ s' \in \gamma^{\varphi}\ s\})\ \lceil Q \rceil = \lceil \lambda s.\ s \in S \longrightarrow (\forall\, t \in T.\ Q\ (\varphi\ t\ s))\rceil$
⟨*proof*⟩

end

Verification with differential invariants

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

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

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

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

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

context *local-flow*
begin

lemma *wp-diff-inv-eq*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows *diff-invariant* $I\ (\lambda t.\ f)\ U\ S\ 0\ (\lambda s.\ True) =$
$(\lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil = wp\ (x' = (\lambda t.\ f)\ \&\ (\lambda s.\ True)\ on\ U\ S\ @\ 0)\ \lceil \lambda s.\ s \in S \longrightarrow I\ s \rceil)$
⟨*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*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil I \rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I)\ \lceil Q \rceil$
⟨*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* ::$((' a::banach) \Rightarrow ' a) \Rightarrow ' a\ pred \Rightarrow ' a\ rel$ $(\langle (1x' =\text{-}\ \&\ \text{-}) \rangle)$
where $(x' = f\ \&\ G) \equiv (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

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

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

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

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t \geq 0.\ (\forall \tau \in \{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$

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

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

lemma *wp-g-evol-IdD*:
assumes $wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil = Id$
and $\forall \tau \in (down\ (U\ s)\ t).\ (s,\ x\ \tau) \in (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$
shows $\forall \tau \in (down\ (U\ s)\ t).\ C\ (x\ \tau)$
⟨*proof*⟩

lemma *diff-cut-axiom*:
assumes $wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil = Id$
shows $wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
⟨*proof*⟩

lemma *diff-cut-rule*:
assumes *wp-C*: $\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil$
and *wp-Q*: $\lceil P \rceil \leq wp\ (x' = f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
⟨*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 $(s,s) \in wp\ (x' = f\ \&\ G)\ \lceil I \rceil$
⟨*proof*⟩

lemma *diff-inv-axiom2*:
assumes *picard-lindeloef* $(\lambda t.\ f)\ UNIV\ UNIV\ 0$
and $\bigwedge s.\ \{t::real.\ t \geq 0\} \subseteq picard\text{-}lindeloef.ex\text{-}ivl\ (\lambda t.\ f)\ UNIV\ UNIV\ 0\ s$
and *diff-invariant* $I\ (\lambda t.\ f)\ (\lambda s.\ \{t::real.\ t \geq 0\})\ UNIV\ 0\ G$
shows $wp\ (x' = f\ \&\ G)\ \lceil I \rceil = wp\ \lceil G \rceil\ \lceil I \rceil$
⟨*proof*⟩

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

⟨*proof*⟩

end

6.4 Examples

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

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

begin

6.4.1 Pendulum

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

abbreviation *fpend* :: *real^2* ⇒ *real^2* (‹*f*›)
where *f s* ≡ (χ *i*. *if i = 1 then s$2 else* −*s$1*)

abbreviation *pend-flow* :: *real* ⇒ *real^2* ⇒ *real^2* (‹φ›)
where φ *t s* ≡ (χ *i*. *if i = 1 then s$1 * cos t + s$2 * sin t*
else − *s$1 * sin t + s$2 * cos t*)

— Verified by providing dynamics.

lemma *pendulum-dyn*:
⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉ ≤ *wp* (*EVOL* φ *G T*) ⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉
⟨*proof*⟩

lemma *pendulum-inv*:
⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉ ≤ *wp* (*x´= f* & *G*) ⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉
⟨*proof*⟩

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

lemma *pendulum-flow*:
⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉ ≤ *wp* (*x´= f* & *G*) ⌈λ*s*. $r^2 = (s\$1)^2 + (s\$2)^2$⌉
⟨*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* ⇒ *real^2* ⇒ *real^2* (‹*f*›)
  where *f g s* ≡ (*χ i. if i = 1 then s$2 else g*)

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real^2* ⇒ *real^2* (‹*φ*›)
  where *φ g t s* ≡ (*χ i. if i = 1 then g * t ^ 2/2 + s$2 * t + s$1 else g * t + s$2*)

— Verified with differential invariants.

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

lemma *inv-imp-pos-le*[*bb-real-arith*]:
  assumes *0 > g* **and** *inv*: *2 * g * x − 2 * g * h = v * v*
  shows (*x*::*real*) ≤ *h*
⟨*proof*⟩

lemma *bouncing-ball-inv*:
  fixes *h*::*real*
  shows *g < 0* ⟹ *h ≥ 0* ⟹ ⌈*λs. s$1 = h ∧ s$2 = 0*⌉ ≤
  wp
    (*LOOP*
      ((*x´= f g &* (*λ s. s$1 ≥ 0*) *DINV* (*λs. 2 * g * s$1 − 2 * g * h − s$2 * s$2 = 0*));
       (*IF* (*λ s. s$1 = 0*) *THEN* (*2 ::= (λs. − s$2)*) *ELSE skip*))
    INV (*λs. 0 ≤ s$1 ∧ 2 * g * s$1 − 2 * g * h − s$2 * s$2 = 0*)
) ⌈*λs. 0 ≤ s$1 ∧ s$1 ≤ h*⌉
  ⟨*proof*⟩

lemma *inv-conserv-at-ground*[*bb-real-arith*]:
  assumes *invar*: *2 * g * x = 2 * g * h + v * v*
    and *pos*: *g * τ^2 / 2 + v * τ + (x::real) = 0*
  shows *2 * g * h + (g * τ * (g * τ + v) + v * (g * τ + v)) = 0*
⟨*proof*⟩

lemma *inv-conserv-at-air*[*bb-real-arith*]:
  assumes *invar*: *2 * g * x = 2 * g * h + v * v*
  shows *2 * g * (g * τ^2 / 2 + v * τ + (x::real)) =*
  *2 * g * h + (g * τ * (g * τ + v) + v * (g * τ + v))* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *bouncing-ball-dyn*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$
$(LOOP$
$((EVOL\ (\varphi\ g)\ (\lambda s.\ 0 \leq s\$1)\ T);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
⟨*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 $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$
$(LOOP$
$((x' = (\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
⟨*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 *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -\ a * (T - U)$ where U is $L \geq 0$ when the heater is on, and *0* when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, *3* is the temperature detected by the thermometer, and *4* states whether the heater is on ($s\$4 = 1$) or off ($s\$4 = 0$). We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

abbreviation *temp-vec-field* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f a L s* ≡ (χ *i*. *if i = 2 then 1 else* (*if i = 1 then − a ∗* (*s$1 − L*) *else 0*))

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

(*if i = 2 then t + s$2 else s$i*))

— Verified with the flow.

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

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*: $a > 0 \Longrightarrow$ *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 $\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$
(*LOOP*
— control
((*2* ::= $(\lambda s.\ 0)$);(*3* ::= $(\lambda s.\ s\$1)$);
(*IF* $(\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)$ *THEN* (*4* ::= $(\lambda s.1)$) *ELSE*
(*IF* $(\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)$ *THEN* (*4* ::= $(\lambda s.0)$) *ELSE skip*));
— dynamics
(*IF* $(\lambda s.\ s\$4 = 0)$ *THEN* (*x´*= *f a 0* & $(\lambda s.\ s\$2 \leq -(ln\ (Tmin/s\$3))/a)$)
ELSE (*x´*= *f a L* & $(\lambda s.\ s\$2 \leq -(ln\ ((L-Tmax)/(L-s\$3)))/a)))$)
INV $(\lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)))$
$\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \rceil$
⟨*proof*⟩

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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f k s* ≡ (χ *i*. *if i = 2 then 1 else* (*if i = 1 then k else 0*))

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

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

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

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

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

lemma *tank-arith*:
assumes *0* ≤ (*τ*::*real*) **and** *0* < c_o **and** $c_o < c_i$
shows ∀*τ*∈{*0*..*τ*}. *τ* ≤ − ((*hmin* − *y*) / c_o) ⟹ *hmin* ≤ *y* − c_o ∗ *τ*
and ∀*τ*∈{*0*..*τ*}. *τ* ≤ (*hmax* − *y*) / (c_i − c_o) ⟹ (c_i − c_o) ∗ *τ* + *y* ≤ *hmax*
and *hmin* ≤ *y* ⟹ *hmin* ≤ (c_i − c_o) ∗ *τ* + *y*
and *y* ≤ *hmax* ⟹ *y* − c_o ∗ *τ* ≤ *hmax*
⟨*proof*⟩

lemma *tank-flow*:
assumes *0* < c_o **and** $c_o < c_i$
shows ⌈λ*s*. *I hmin hmax s*⌉ ≤ *wp*
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s*. *s$1*));
(*IF* (λ*s*. *s$4* = *0* ∧ *s$3* ≤ *hmin* + *1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s*. *s$4* = *1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));

— dynamics
(*IF* (λ*s*. *s*$4 = 0) *THEN* (*x´* = *f* (*c*$_i$−*c*$_o$) & (*G hmax* (*c*$_i$−*c*$_o$)))
ELSE (*x´* = *f* (−*c*$_o$) & (*G hmin* (−*c*$_o$))))) *INV I hmin hmax*)
⌈λ*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*: (⋀*x*. (*f*$_\bullet$) *x* = (*g*$_\bullet$) *x*) ⟹ *f* = *g*
⟨*proof*⟩

lemma *nd-fun-eq-iff*: (*f* = *g*) = (∀ *x*. (*f*$_\bullet$) *x* = (*g*$_\bullet$) *x*)
⟨*proof*⟩

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

definition *0* = ζ$^\bullet$

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

definition *f* + *g* = ((*f*$_\bullet$) ⊔ (*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-zerol*[*nd-fun-ka*]: $0 + x = x$
and *nd-fun-mult-zerol*[*nd-fun-ka*]: $0 \cdot x = 0$
and *nd-fun-mult-zeror*[*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*]: $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
and *nd-star-unfoldr*[*nd-fun-ka*]: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$ **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* = (λ*s*. *if* ((*f*$_\bullet$) *s* = {}) *then* {*s*} *else* {})$^\bullet$

lemma *nd-fun-ad-zero*[*nd-fun-ka*]: *ad x* · *x* = *0*
  and *nd-fun-ad*[*nd-fun-ka*]: *ad* (*x* · *y*) + *ad* (*x* · *ad* (*ad y*)) = *ad* (*x* · *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* ⇒ *bool*

notation *fbox* (‹*wp*›)

unbundle *no floor-ceiling-syntax*
no-notation *Relation.relcomp* (**infixl** ‹;› *75*)
  and *Range-Semiring.antirange-semiring-class.ars-r* (‹*r*›)
  and *antidomain-semiringl.ads-d* (‹*d*›)

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

lemma *p2ndf-simps*[*simp*]:
  ⌈*P*⌉ ≤ ⌈*Q*⌉ = (∀ *s*. *P s* ⟶ *Q s*)
  (⌈*P*⌉ = ⌈*Q*⌉) = (∀ *s*. *P s* = *Q s*)
  (⌈*P*⌉ · ⌈*Q*⌉) = ⌈λ *s*. *P s* ∧ *Q s*⌉
  (⌈*P*⌉ + ⌈*Q*⌉) = ⌈λ *s*. *P s* ∨ *Q s*⌉
  ad ⌈*P*⌉ = ⌈λ*s*. ¬ *P s*⌉
  d ⌈*P*⌉ = ⌈*P*⌉ ⌈*P*⌉ ≤ *η*$^\bullet$
  ⟨*proof*⟩

Lemmas for verification condition generation

lemma *wp-nd-fun*: *wp F* ⌈*P*⌉ = ⌈λ*s*. ∀ *s′*. *s′* ∈ ((*F*$_\bullet$) *s*) ⟶ *P s′*⌉
  ⟨*proof*⟩

abbreviation *skip* :: *′a nd-fun*

where *skip* ≡ *1*

— Tests

lemma *wp-test*[*simp*]: *wp* ⌈*P*⌉ ⌈*Q*⌉ = ⌈λ*s*. *P s* ⟶ *Q s*⌉
⟨*proof*⟩

definition *assign* :: *'b* ⇒ (*'a*⁀*'b* ⇒ *'a*) ⇒ (*'a*⁀*'b*) *nd-fun* (‹(*2*- ::= -)› [*70*, *65*] *61*)
where (*x* ::= *e*) = (λ*s*. {*vec-upd s x* (*e s*)})•

lemma *wp-assign*[*simp*]: *wp* (*x* ::= *e*) ⌈*Q*⌉ = ⌈λ*s*. *Q* (χ *j*. (((*$*) *s*)(*x* := (*e s*))) *j*)⌉
⟨*proof*⟩

definition *nondet-assign* :: *'b* ⇒ (*'a*⁀*'b*) *nd-fun* (‹(*2*- ::= *?*)› [*70*] *61*)
where (*x* ::= *?*) = (λ*s*. {(*vec-upd s x k*)|*k*. *True*})•

lemma *wp-nondet-assign*[*simp*]: *wp* (*x* ::= *?*) ⌈*P*⌉ = ⌈λ*s*. ∀ *k*. *P* (χ *j*. (((*$*) *s*)(*x* := *k*)) *j*)⌉
⟨*proof*⟩

lemma *le-wp-choice-iff*: ⌈*P*⌉ ≤ *wp* (*X* + *Y*) ⌈*Q*⌉ ⟷ ⌈*P*⌉ ≤ *wp X* ⌈*Q*⌉ ∧ ⌈*P*⌉ ≤ *wp Y* ⌈*Q*⌉
⟨*proof*⟩

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

— Conditional statement

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

— Finite iteration

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

lemma *change-loopI*: *LOOP X INV G* = *LOOP X INV I*
⟨*proof*⟩

lemma *wp-loopI*: ⌈*P*⌉ ≤ ⌈*I*⌉ ⟹ ⌈*I*⌉ ≤ ⌈*Q*⌉ ⟹ ⌈*I*⌉ ≤ *wp R* ⌈*I*⌉ ⟹ ⌈*P*⌉ ≤ *wp* (*LOOP R INV I*) ⌈*Q*⌉
⟨*proof*⟩

lemma *wp-loopI-break*:
⌈*P*⌉ ≤ *wp Y* ⌈*I*⌉ ⟹ ⌈*I*⌉ ≤ *wp X* ⌈*I*⌉ ⟹ ⌈*I*⌉ ≤ ⌈*Q*⌉ ⟹ ⌈*P*⌉ ≤ *wp* (*Y* ; (*LOOP X INV I*)) ⌈*Q*⌉

⟨*proof*⟩

6.6.2 Evolution commands

Verification by providing evolution

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

lemma *wp-g-dyn*[*simp*]:
  fixes φ :: ('*a*::*preorder*) ⇒ '*b* ⇒ '*b*
  shows *wp* (*EVOL* φ *G U*) ⌈*Q*⌉ = ⌈λ*s*. ∀ *t*∈*U s*. (∀ τ∈*down* (*U s*) *t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*)⌉
  ⟨*proof*⟩

Verification by providing solutions

definition *g-ode* ::(*real* ⇒ ('*a*::*banach*) ⇒ '*a*) ⇒ '*a pred* ⇒ ('*a* ⇒ *real set*) ⇒ '*a set* ⇒
  real ⇒ '*a nd-fun* (‹(*1x´*= - & - *on* - - @ -)›)
  where (*x´*= *f* & *G on U S* @ t_0) ≡ (λ *s*. *g-orbital f G U S* t_0 *s*)•

lemma *wp-g-orbital*: *wp* (*x´*= *f* & *G on U S* @ t_0) ⌈*Q*⌉=
  ⌈λ*s*. ∀ *X*∈*Sols f U S* t_0 *s*. ∀ *t*∈*U s*. (∀ τ∈*down* (*U s*) *t*. *G* (*X* τ)) ⟶ *Q* (*X t*)⌉
  ⟨*proof*⟩

context *local-flow*
begin

lemma *wp-g-ode-subset*:
  assumes ⋀*s*. *s* ∈ *S* ⟹ *0* ∈ *U s* ∧ *is-interval* (*U s*) ∧ *U s* ⊆ *T*
  shows *wp* (*x´*= (λ*t*. *f*) & *G on U S* @ *0*) ⌈*Q*⌉ =
  ⌈λ*s*. *s* ∈ *S* ⟶ (∀ *t*∈*U s*. (∀ τ∈*down* (*U s*) *t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))⌉
  ⟨*proof*⟩

lemma *wp-g-ode*: *wp* (*x´*= (λ*t*. *f*) & *G on* (λ*s*. *T*) *S* @ *0*) ⌈*Q*⌉ =
  ⌈λ*s*. *s* ∈ *S* ⟶ (∀ *t*∈*T*. (∀ τ∈*down T t*. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))⌉
  ⟨*proof*⟩

lemma *wp-g-ode-ivl*: *t* ≥ *0* ⟹ *t* ∈ *T* ⟹ *wp* (*x´*= (λ*t*. *f*) & *G on* (λ*s*. {*0*..*t*}) *S* @ *0*) ⌈*Q*⌉ =
  ⌈λ*s*. *s* ∈ *S* ⟶ (∀ *t*∈{*0*..*t*}. (∀ τ∈{*0*..*t*}. *G* (φ τ *s*)) ⟶ *Q* (φ *t s*))⌉
  ⟨*proof*⟩

lemma *wp-orbit*: *wp* (γ^{φ}•) ⌈*Q*⌉ = ⌈λ *s*. *s* ∈ *S* ⟶ (∀ *t* ∈ *T*. *Q* (φ *t s*))⌉
  ⟨*proof*⟩

end

Verification with differential invariants

definition *g-ode-inv* :: (*real* ⇒ ('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ ('*a* ⇒ *real set*) ⇒ '*a set* ⇒
real ⇒ '*a pred* ⇒ '*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* = (*λs. G s ∧ 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*⌉ ≤ ⌈*I*⌉ **and** ⌈*I*⌉ ≤ *wp* (*x´= f & G on U S @ t*$_0$) ⌈*I*⌉ **and** ⌈*I*⌉ ≤ ⌈*Q*⌉
shows ⌈*P*⌉ ≤ *wp* (*x´= f & G on U S @ t*$_0$) ⌈*Q*⌉
⟨*proof*⟩

lemma *wp-diff-inv*[*simp*]: (⌈*I*⌉ ≤ *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*⌉ ≤ *wp* (*x´= f & (λs. True) on U S @ t*$_0$) ⌈*I*⌉
shows ⌈*I*⌉ ≤ *wp* (*x´= f & G on U S @ t*$_0$) ⌈*I*⌉
⟨*proof*⟩

context *local-flow*
begin

lemma *wp-diff-inv-eq*:
assumes ⋀*s. s* ∈ *S* ⟹ *0* ∈ *U s* ∧ *is-interval* (*U s*) ∧ *U s* ⊆ *T*
shows *diff-invariant I* (*λt. f*) *U S 0* (*λs. True*) =
(⌈*λs. s* ∈ *S* ⟶ *I s*⌉ = *wp* (*x´= (λt. f) & (λs. True) on U S @ 0*) ⌈*λs. s* ∈ *S* ⟶ *I s*⌉)
⟨*proof*⟩

lemma *diff-inv-eq-inv-set*:
diff-invariant I (*λt. f*) (*λs. T*) *S 0* (*λs. True*) = (∀ *s. I s* ⟶ *γ*$^{\varphi}$ *s* ⊆ {*s. I s*})
⟨*proof*⟩

end

lemma *wp-g-odei*: ⌈*P*⌉ ≤ ⌈*I*⌉ ⟹ ⌈*I*⌉ ≤ *wp* (*x´= f & G on U S @ t*$_0$) ⌈*I*⌉ ⟹ ⌈*λs. I s ∧ G s*⌉ ≤ ⌈*Q*⌉ ⟹
⌈*P*⌉ ≤ *wp* (*x´= f & G on U S @ t*$_0$ *DINV I*) ⌈*Q*⌉
⟨*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::*banach*)⇒'a) ⇒ 'a *pred* ⇒ 'a *nd-fun* (‹(*1x´=- & -*)›)

where $(x´= f\ \&\ G) \equiv (x´= (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{t.\ t \geq 0\})\ UNIV\ @\ 0)$

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

lemma *diff-solve-axiom1*:
assumes *local-flow f UNIV UNIV* φ
shows *wp* $(x´= f\ \&\ G)\ \lceil Q \rceil =$
$\lceil \lambda s.\ \forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil$
⟨*proof*⟩

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

lemma *diff-solve-rule*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P \rceil \leq wp\ (x´= f\ \&\ G)\ \lceil Q \rceil$
⟨*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 $\lceil P \rceil \leq wp\ (x´= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
⟨*proof*⟩

lemma *wp-g-orbit-IdD*:
assumes $wp\ (x´= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil = \eta^\bullet$
and $\forall \tau{\in}(down\ (U\ s)\ t).\ x\ \tau \in g\text{-}orbital\ f\ G\ U\ S\ t_0\ s$
shows $\forall \tau{\in}(down\ (U\ s)\ t).\ C\ (x\ \tau)$
⟨*proof*⟩

lemma *diff-cut-axiom*:
assumes *wp* $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ \lceil C \rceil = \eta^\bullet$
shows *wp* $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ \lceil Q \rceil =$ *wp* $(x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \ on \ U \ S \ @ \ t_0) \ \lceil Q \rceil$
⟨*proof*⟩

lemma *diff-cut-rule*:
assumes *wp-C*: $\lceil P \rceil \leq$ *wp* $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ \lceil C \rceil$
and *wp-Q*: $\lceil P \rceil \leq$ *wp* $(x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \ on \ U \ S \ @ \ t_0) \ \lceil Q \rceil$
shows $\lceil P \rceil \leq$ *wp* $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ \lceil Q \rceil$
⟨*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 $s \in ((wp \ (x' = f \ \& \ G) \ \lceil I \rceil)_\bullet) \ s$
⟨*proof*⟩

lemma *diff-inv-axiom2*:
assumes *picard-lindeloef* $(\lambda t. \ f) \ UNIV \ UNIV \ 0$
and $\bigwedge s. \ \{t::real. \ t \geq 0\} \subseteq$ *picard-lindeloef.ex-ivl* $(\lambda t. \ f) \ UNIV \ UNIV \ 0 \ s$
and *diff-invariant* $I \ (\lambda t. \ f) \ (\lambda s. \ \{t::real. \ t \geq 0\}) \ UNIV \ 0 \ G$
shows *wp* $(x' = f \ \& \ G) \ \lceil I \rceil =$ *wp* $\lceil G \rceil \ \lceil I \rceil$
⟨*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 $\lceil P \rceil \leq$ *wp* $(x' = f \ \& \ G \ on \ U \ S \ @ \ t_0) \ \lceil Q \rceil$
⟨*proof*⟩

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 ⇒ real^2* (‹*f*›)
  where *f s* ≡ (*χ i. if i = 1 then s$2 else −s$1*)

abbreviation *pend-flow* :: *real ⇒ real^2 ⇒ real^2* (‹*φ*›)
  where *φ t s* ≡ (*χ i. if i = 1 then s$1 ∗ cos t + s$2 ∗ sin t*
  else − s$1 ∗ sin t + s$2 ∗ cos t)

— Verified by providing dynamics.

lemma *pendulum-dyn*:
  ⌈*λs. r*^{2} = (*s$1*)^2 + (*s$2*)^2⌉ ≤ *wp* (*EVOL φ G T*) ⌈*λs. r*^{2} = (*s$1*)^2 + (*s$2*)^2⌉
  ⟨*proof*⟩

lemma *pendulum-inv*:
  ⌈*λs. r*^{2} = (*s$1*)^2 + (*s$2*)^2⌉ ≤ *wp* (*x´= f & G*) ⌈*λs. r*^{2} = (*s$1*)^2 + (*s$2*)^2⌉
  ⟨*proof*⟩

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

lemma *pendulum-flow*:
  ⌈*λs. r*^{2} = (*s$1*)^2 + (*s$2*)^2⌉ ≤ *wp* (*x´= f & G*) ⌈*λs. 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 ⇒ real^2 ⇒ real^2* (‹*f*›)
  where *f g s* ≡ (*χ i. if i = 1 then s$2 else g*)

abbreviation *ball-flow* :: *real ⇒ real ⇒ real^2 ⇒ real^2* (‹*φ*›)
  where *φ g t s* ≡ (*χ i. if i = 1 then g ∗ t ^ 2/2 + s$2 ∗ t + s$1 else g ∗ t +*
s$2)

— Verified with differential invariants.

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

lemma *inv-imp-pos-le*[*bb-real-arith*]:
assumes $0 > g$ **and** *inv*: $2 * g * x - 2 * g * h = v * v$
shows $(x::real) \leq h$
⟨*proof*⟩

lemma *bouncing-ball-inv*:
fixes $h::real$
shows $g < 0 \Longrightarrow h \geq 0 \Longrightarrow \lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq$
wp
(*LOOP*
((*x´*= *f g* & (λ *s*. *s*$1 ≥ 0) *DINV* (λ*s*. 2 ∗ *g* ∗ *s*$1 − 2 ∗ *g* ∗ *h* − *s*$2 ∗ *s*$2
= 0));
(*IF* (λ *s*. *s*$1 = 0) *THEN* (2 ::= (λ*s*. − *s*$2)) *ELSE skip*))
INV (λ*s*. 0 ≤ *s*$1 ∧ 2 ∗ *g* ∗ *s*$1 − 2 ∗ *g* ∗ *h* − *s*$2 ∗ *s*$2 = 0)
) ⌈λ*s*. 0 ≤ *s*$1 ∧ *s*$1 ≤ *h*⌉
⟨*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 $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$
(*LOOP*
((*EVOL* (φ *g*) (λ*s*. 0 ≤ *s*$1) *T*);
(*IF* (λ *s*. *s*$1 = 0) *THEN* (2 ::= (λ*s*. − *s*$2)) *ELSE skip*))
INV (λ*s*. 0 ≤ *s*$1 ∧ 2 ∗ *g* ∗ *s*$1 = 2 ∗ *g* ∗ *h* + *s*$2 ∗ *s*$2))
⌈λ*s*. 0 ≤ *s*$1 ∧ *s*$1 ≤ *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 $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$

$(LOOP$
$((x' = (\lambda t.\ f\ g)\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ on\ (\lambda s.\ UNIV)\ UNIV\ @\ 0);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
⟨*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* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f a L s* ≡ (χ *i. if i = 2 then 1 else* (*if i = 1 then* − *a* ∗ (*s*$1 − *L*) *else 0*))

abbreviation *temp-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹φ›)
where φ *a L t s* ≡ (χ *i. if i = 1 then* − *exp*(−*a* ∗ *t*) ∗ (*L* − *s*$1) + *L else* (*if i = 2 then t* + *s*$2 *else s*$*i*))

— Verified with the flow.

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

lemma *local-lipschitz-temp-dyn*:
assumes *0* < (*a*::*real*)
shows *local-lipschitz UNIV UNIV* (λ*t*::*real. f a L*)
⟨*proof*⟩

lemma *local-flow-temp*: *a* > *0* ⟹ *local-flow* (*f a L*) *UNIV UNIV* (φ *a L*)
⟨*proof*⟩

lemma *temp-dyn-down-real-arith*:
assumes *a* > *0* **and** *Thyps*: *0* < *Tmin Tmin* ≤ *T T* ≤ *Tmax*
and *thyps*: *0* ≤ (*t*::*real*) ∀τ∈{*0*..*t*}. $\tau \leq -$ (*ln* (*Tmin* / *T*) / *a*)
shows *Tmin* ≤ *exp* (−*a* ∗ *t*) ∗ *T* **and** *exp* (−*a* ∗ *t*) ∗ *T* ≤ *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 $\lceil \lambda s.\ Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$
  (*LOOP*
    — control
    ((*2* ::= (λ*s*. *0*));(*3* ::= (λ*s*. *s*$*1*));
    (*IF* (λ*s*. *s*$*4* = *0* ∧ *s*$*3* ≤ *Tmin* + *1*) *THEN* (*4* ::= (λ*s*.*1*)) *ELSE*
    (*IF* (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *Tmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*));
    — dynamics
    (*IF* (λ*s*. *s*$*4* = *0*) *THEN* (*x´*= *f a 0* & (λ*s*. *s*$*2* ≤ − (*ln* (*Tmin*/*s*$*3*))/*a*))
    ELSE (*x´*= *f a L* & (λ*s*. *s*$*2* ≤ − (*ln* ((*L*−*Tmax*)/(*L*−*s*$*3*)))/*a*))))
  INV (λ*s*. *Tmin* ≤*s*$*1* ∧ *s*$*1* ≤ *Tmax* ∧ (*s*$*4* = *0* ∨ *s*$*4* = *1*)))
  ⌈λ*s*. *Tmin* ≤ *s*$*1* ∧ *s*$*1* ≤ *Tmax*⌉
  ⟨*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 $hmin \leq h \leq hmax$. 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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
  where *f k s* ≡ (χ *i*. *if i* = *2 then 1 else* (*if i* = *1 then k else 0*))

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

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

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

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

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

lemma *tank-arith*:
assumes *0* ≤ (*τ*::*real*) **and** *0* < c_o **and** c_o < c_i
shows ∀*τ*∈{*0*..*τ*}. *τ* ≤ − ((*hmin* − *y*) / c_o) ⟹ *hmin* ≤ *y* − c_o ∗ *τ*
and ∀*τ*∈{*0*..*τ*}. *τ* ≤ (*hmax* − *y*) / (c_i − c_o) ⟹ (c_i − c_o) ∗ *τ* + *y* ≤ *hmax*
and *hmin* ≤ *y* ⟹ *hmin* ≤ (c_i − c_o) ∗ *τ* + *y*
and *y* ≤ *hmax* ⟹ *y* − c_o ∗ *τ* ≤ *hmax*
⟨*proof*⟩

lemma *tank-flow*:
assumes *0* < c_o **and** c_o < c_i
shows ⌈λ*s*. *I hmin hmax s*⌉ ≤ *wp*
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s*. *s$1*));
(*IF* (λ*s*. *s$4* = *0* ∧ *s$3* ≤ *hmin* + *1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s*. *s$4* = *1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s*. *s$4* = *0*) *THEN* (*x´*= *f* (c_i−c_o) & (*G hmax* (c_i−c_o)))
ELSE (*x´*= *f* (−c_o) & (*G hmin* (−c_o))))) *INV I hmin hmax*)
⌈λ*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

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* (‹(‹*notation*=‹*mixfix if expression*››*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* ⇒ *'a* ⇒ *'a* ⇒ *bool* (‹*H*›) **where**
H p x q ⟷ tt *p* · *x* ≤ *x* · tt *q*

lemma *H-consl*: tt *p* ≤ tt *p'* ⟹ *H p' x q* ⟹ *H p x q*
⟨*proof*⟩

lemma *H-consr*: tt *q'* ≤ tt *q* ⟹ *H p x q'* ⟹ *H p x q*
⟨*proof*⟩

lemma *H-cons*: tt *p* ≤ tt *p'* ⟹ tt *q'* ≤ tt *q* ⟹ *H p' x q'* ⟹ *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* ⟹ *H r y q* ⟹ *H p* (*x* · *y*) *q*
⟨*proof*⟩

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

definition *kat-cond* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'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* ⟷ tt *p* · *x* · *n q* = *0*

⟨*proof*⟩

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

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

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

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

lemma *H-exp1*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ q \Longrightarrow H\ p\ (\mathfrak{tt}\ r \cdot x)\ q$
⟨*proof*⟩

lemma *H-while*: $H\ (\mathfrak{tt}\ p \cdot \mathfrak{tt}\ r)\ x\ p \Longrightarrow H\ p\ (\mathit{while}\ r\ \mathit{do}\ x)\ (\mathfrak{tt}\ p \cdot n\ r)$
⟨*proof*⟩

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

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

lemma *H-star-inv*:
assumes $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i$ **and** $H\ i\ x\ i$ **and** $(\mathfrak{tt}\ i) \leq (\mathfrak{tt}\ q)$
shows $H\ p\ (x^\star)\ q$
⟨*proof*⟩

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

lemma *H-loop*: $H\ p\ x\ p \Longrightarrow H\ p\ (\mathit{loop}\ x\ \mathit{inv}\ i)\ p$
⟨*proof*⟩

lemma *H-loop-inv*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i \Longrightarrow H\ i\ x\ i \Longrightarrow \mathfrak{tt}\ i \leq \mathfrak{tt}\ q \Longrightarrow H\ p\ (\mathit{loop}\ x\ \mathit{inv}\ i)\ q$
⟨*proof*⟩

lemma *H-inv*: $\mathfrak{tt}\ p \leq \mathfrak{tt}\ i \Longrightarrow \mathfrak{tt}\ i \leq \mathfrak{tt}\ q \Longrightarrow H\ i\ x\ i \Longrightarrow H\ p\ x\ q$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *H-inv-mult*: tt $i = i \Longrightarrow$ tt $j = j \Longrightarrow H\ i\ x\ i \Longrightarrow H\ j\ x\ j \Longrightarrow H\ (i \cdot j)\ x\ (i \cdot j)$
⟨*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*: $x \leq Ref\ p\ q \longleftrightarrow H\ p\ x\ q$

begin

lemma *R1*: $H\ p\ (Ref\ p\ q)\ q$
⟨*proof*⟩

lemma *R2*: $H\ p\ x\ q \Longrightarrow x \leq Ref\ p\ q$
⟨*proof*⟩

lemma *R-cons*: tt $p \leq$ tt $p' \Longrightarrow$ tt $q' \leq$ tt $q \Longrightarrow Ref\ p'\ q' \leq Ref\ p\ q$
⟨*proof*⟩

lemma *R-skip*: $1 \leq Ref\ p\ p$
⟨*proof*⟩

lemma *R-zero-one*: $x \leq Ref\ 0\ 1$
⟨*proof*⟩

lemma *R-one-zero*: $Ref\ 1\ 0 = 0$
⟨*proof*⟩

lemma *R-abort*: $0 \leq Ref\ p\ q$
⟨*proof*⟩

lemma *R-seq*: $(Ref\ p\ r) \cdot (Ref\ r\ q) \leq Ref\ p\ q$
⟨*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*: $(Ref\ i\ i)^\star \leq Ref\ i\ i$
⟨*proof*⟩

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

lemma *R-inv*: $tt\ p \leq tt\ i \Longrightarrow tt\ i \leq tt\ q \Longrightarrow Ref\ i\ i \leq Ref\ p\ q$
⟨*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**
$rel\text{-}R\ P\ Q = \bigcup \{X.\ rel\text{-}kat.Hoare\ P\ X\ Q\}$

interpretation *rel-rkat*: *rkat* $(\cup)\ (O)\ Id\ \{\}\ (\subseteq)\ (\subset)\ rtrancl\ (\lambda X.\ Id \cap -X)\ rel\text{-}R$
⟨*proof*⟩

7.3.1 Regular programs

Lemmas for manipulation of predicates in the relational model

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

notation *Id* (‹*skip*›)
and *empty* (‹*abort*›)
and *relcomp* (**infixr** ‹;› *75*)

unbundle *no floor-ceiling-syntax*
no-notation *tau* (‹τ›)
and *n-op* (‹*n* -› [*90*] *91*)

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

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

Lemmas for verification condition generation

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

abbreviation *relHoare* (‹**{-}**-**{-}**›)
where **{***P***}***X***{***Q***}** ≡ *rel-kat.Hoare* ⌈*P*⌉ *X* ⌈*Q*⌉

lemma *rel-kat-H*: **{***P***}** *X* **{***Q***}** ⟷ (∀ *s s'*. *P s* ⟶ (*s*,*s'*) ∈ *X* ⟶ *Q s'*)
⟨*proof*⟩

lemma *sH-skip*[*simp*]: **{***P***}** *skip* **{***Q***}** ⟷ ⌈*P*⌉ ≤ ⌈*Q*⌉
⟨*proof*⟩

lemma *H-skip*: **{***P***}** *skip* **{***P***}**
⟨*proof*⟩

lemma *sH-test*[*simp*]: **{***P***}** ⌈*R*⌉ **{***Q***}** = (∀ *s*. *P s* ⟶ *R s* ⟶ *Q s*)
⟨*proof*⟩

lemma *sH-abort*[*simp*]: **{***P***}** *abort* **{***Q***}** ⟷ *True*
⟨*proof*⟩

lemma *H-abort*: **{***P***}** *abort* **{***Q***}**
⟨*proof*⟩

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

lemma *sH-assign*[*simp*]: **{***P***}** (*x* ::= *e*) **{***Q***}** ⟷ (∀ *s*. *P s* ⟶ *Q* (χ *j*. ((($) *s*)(*x* := (*e s*))) *j*))
⟨*proof*⟩

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

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

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*⟩

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

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

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

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

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

abbreviation *cond-sugar* :: *'a pred* ⇒ *'a rel* ⇒ *'a rel* ⇒ *'a rel*
(‹*IF - THEN - ELSE* -› [*64*,*64*] *63*)
where *IF B THEN X ELSE Y* ≡ *rel-kat.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* ⇒ *'a pred* ⇒ *'a rel* ⇒ *'a rel*
(‹*WHILE - INV - DO* -› [*64*,*64*,*64*] *63*)
where *WHILE B INV I DO X* ≡ *rel-kat.kat-while-inv* ⌈*B*⌉ ⌈*I*⌉ *X*

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

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

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

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

lemma *H-loopI*: $\{I\}\ X\ \{I\} \Longrightarrow \lceil P \rceil \subseteq \lceil I \rceil \Longrightarrow \lceil I \rceil \subseteq \lceil Q \rceil \Longrightarrow \{P\}\ (LOOP\ X\ INV\ I)\ \{Q\}$
⟨*proof*⟩

7.3.2 Evolution commands

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

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

lemma *H-g-evol*:
fixes φ :: ('*a*::*preorder*) ⇒ *'b* ⇒ *'b*
assumes $P = (\lambda s.\ (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows $\{P\}\ (EVOL\ \varphi\ G\ U)\ \{Q\}$
⟨*proof*⟩

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

lemma *H-g-orbital*:
$P = (\lambda s.\ (\forall\, X{\in}ivp\text{-}sols\ f\ U\ S\ t_0\ s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \Longrightarrow$
$\{P\}\ (x´= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{Q\}$
⟨*proof*⟩

lemma *sH-g-orbital*: **{**P**}** ($x´ = f$ & G *on* $U\ S$ @ t_0) **{**Q**}** =
($\forall\, s.\ P\ s \longrightarrow (\forall\, X{\in}$*ivp-sols* $f\ U\ S\ t_0\ s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}$*down* $(U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)))$
⟨*proof*⟩

context *local-flow*
begin

lemma *sH-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows **{**P**}** ($x´ = (\lambda t.\ f)$ & G *on* $U\ S$ @ 0) **{**Q**}** =
($\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}$*down* $(U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

lemma *H-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ s \in S \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}$*down* $(U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows **{**P**}** ($x´ = (\lambda t.\ f)$ & G *on* $U\ S$ @ 0) **{**Q**}**
⟨*proof*⟩

lemma *sH-g-ode*: **{**P**}** ($x´ = (\lambda t.\ f)$ & G *on* $(\lambda s.\ T)\ S$ @ 0) **{**Q**}** =
($\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}$*down* $T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

lemma *sH-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow$ **{**P**}** ($x´ = (\lambda t.\ f)$ & G *on* $(\lambda s.\ \{0..t\})\ S$ @ 0) **{**Q**}** =
($\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}\{0..t\}.\ (\forall\, \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

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

end

— Verification with differential invariants

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

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

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

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

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

7.4 Refinement Components

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

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

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

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

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

— Nondeterministic Choice

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

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

lemma *R-choice-mono*: $P' \subseteq P \Longrightarrow Q' \subseteq Q \Longrightarrow P' \cup Q' \subseteq P \cup Q$
⟨*proof*⟩

lemma *R-assign*: $(x ::= e) \leq rel\text{-}R\ \lceil \lambda s.\ P\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j) \rceil\ \lceil P \rceil$
⟨*proof*⟩

lemma *R-assign-law*:
$(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \leq rel\text{-}R\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-assignl*: $P = (\lambda s.\ R\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow$
$(x ::= e)\ ;\ \textit{rel-R}\ \lceil R \rceil\ \lceil Q \rceil \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-assignr*: $R = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow$
$\textit{rel-R}\ \lceil P \rceil\ \lceil R \rceil;\ (x ::= e) \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma $(x ::= e)\ ;\ \textit{rel-R}\ \lceil Q \rceil\ \lceil Q \rceil \leq \textit{rel-R}\ \lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma $\textit{rel-R}\ \lceil Q \rceil\ \lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil;\ (x ::= e) \leq \textit{rel-R}\ \lceil Q \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-nondet-assign*: $(x ::= ?) \leq \textit{rel-R}\ \lceil \lambda s.\ \forall k.\ P\ (\chi\ j.\ (((\$)\ s)(x := k))\ j) \rceil\ \lceil P \rceil$
⟨*proof*⟩

lemma *R-nondet-assign-law*:
$(\forall s.\ P\ s \longrightarrow (\forall k.\ Q\ (\chi\ j.\ (((\$)\ s)(x := k))\ j))) \Longrightarrow (x ::= ?) \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-cond*:
$(\textit{IF}\ B\ \textit{THEN}\ \textit{rel-R}\ \lceil \lambda s.\ B\ s \wedge P\ s \rceil\ \lceil Q \rceil\ \textit{ELSE}\ \textit{rel-R}\ \lceil \lambda s.\ \neg\ B\ s \wedge P\ s \rceil\ \lceil Q \rceil) \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-cond-mono*: $X \leq X' \Longrightarrow Y \leq Y' \Longrightarrow (\textit{IF}\ P\ \textit{THEN}\ X\ \textit{ELSE}\ Y) \leq \textit{IF}\ P\ \textit{THEN}\ X'\ \textit{ELSE}\ Y'$
⟨*proof*⟩

lemma *R-cond-law*: $X \leq \textit{rel-R}\ \lceil \lambda s.\ B\ s \wedge P\ s \rceil\ \lceil Q \rceil \Longrightarrow Y \leq \textit{rel-R}\ \lceil \lambda s.\ \neg\ B\ s \wedge P\ s \rceil\ \lceil Q \rceil \Longrightarrow$
$(\textit{IF}\ B\ \textit{THEN}\ X\ \textit{ELSE}\ Y) \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-while*: $K = (\lambda s.\ P\ s \wedge \neg\ B\ s) \Longrightarrow$
$\textit{WHILE}\ B\ \textit{INV}\ I\ \textit{DO}\ (\textit{rel-R}\ \lceil \lambda s.\ P\ s \wedge B\ s \rceil\ \lceil P \rceil) \leq \textit{rel-R}\ \lceil P \rceil\ \lceil K \rceil$
⟨*proof*⟩

lemma *R-whileI*:
$X \leq \textit{rel-R}\ \lceil I \rceil\ \lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil \lambda s.\ I\ s \wedge B\ s \rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge \neg\ B\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$\textit{WHILE}\ B\ \textit{INV}\ I\ \textit{DO}\ X \leq \textit{rel-R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-while-mono*: $X \leq X' \Longrightarrow (\textit{WHILE}\ P\ \textit{INV}\ I\ \textit{DO}\ X) \subseteq \textit{WHILE}\ P\ \textit{INV}\ I\ \textit{DO}\ X'$

⟨*proof*⟩

lemma *R-while-law*: $X \leq \mathit{rel\text{-}R}\ \lceil \lambda s.\ P\ s \wedge B\ s \rceil\ \lceil P \rceil \Longrightarrow Q = (\lambda s.\ P\ s \wedge \neg\ B\ s) \Longrightarrow$
$(\mathit{WHILE}\ B\ \mathit{INV}\ I\ \mathit{DO}\ X) \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-loop*:$\mathit{LOOP}\ \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil P \rceil\ \mathit{INV}\ I \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil P \rceil$
⟨*proof*⟩

lemma *R-loopI*:
$X \leq \mathit{rel\text{-}R}\ \lceil I \rceil\ \lceil I \rceil \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow \mathit{LOOP}\ X\ \mathit{INV}\ I \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-loop-mono*: $X \leq X' \Longrightarrow \mathit{LOOP}\ X\ \mathit{INV}\ I \subseteq \mathit{LOOP}\ X'\ \mathit{INV}\ I$
⟨*proof*⟩

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

lemma *R-g-evol-law*:
fixes $\varphi :: ('a::\mathit{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall\, s.\ P\ s \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (\mathit{EVOL}\ \varphi\ G\ U) \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-evoll*:
fixes $\varphi :: ('a::\mathit{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(\mathit{EVOL}\ \varphi\ G\ U)\ ;\ \mathit{rel\text{-}R}\ \lceil R \rceil\ \lceil Q \rceil \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-evolr*:
fixes $\varphi :: ('a::\mathit{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$\mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil R \rceil;\ (\mathit{EVOL}\ \varphi\ G\ U) \leq \mathit{rel\text{-}R}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma
fixes $\varphi :: ('a::\mathit{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\mathit{EVOL}\ \varphi\ G\ U\ ;\ \mathit{rel\text{-}R}\ \lceil Q \rceil\ \lceil Q \rceil \leq \mathit{rel\text{-}R}\ \lceil \lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows *rel-R* $\lceil Q \rceil$ $\lceil \lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil$;
EVOL $\varphi\ G\ U \leq$ *rel-R* $\lceil Q \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

context *local-flow*
begin

lemma *R-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq$
rel-R $\lceil \lambda s.\ s{\in}S \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)) \rceil$ $\lceil P \rceil$
⟨*proof*⟩

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

lemma *R-g-odel-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s))$
shows $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)$; *rel-R* $\lceil R \rceil$ $\lceil Q \rceil \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-oder-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $R = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows *rel-R* $\lceil P \rceil$ $\lceil R \rceil$; $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-ode*: $(x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq$
rel-R $\lceil \lambda s.\ s{\in}S \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)) \rceil$ $\lceil P \rceil$
⟨*proof*⟩

lemma *R-g-ode-law*: $(\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-odel*: $P = (\lambda s.\ \forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)$; *rel-R* $\lceil R \rceil$ $\lceil Q \rceil \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-oder*: $R = (\lambda s.\ \forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
rel-R $\lceil P \rceil$ $\lceil R \rceil$; $(x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0) \leq$ *rel-R* $\lceil P \rceil$ $\lceil Q \rceil$
⟨*proof*⟩

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

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant I f T S* t_0 $G \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$(x' = f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) \leq rel\text{-}R\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

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

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

lemma *diff-solve-rule1*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows **{***P***}** $(x' = f\ \&\ G)$ **{***Q***}**
⟨*proof*⟩

lemma *diff-solve-rule2*:
fixes *c*::'a::{*heine-borel*, *banach*}
assumes $\forall s.\ P\ s \longrightarrow (\forall t{\geq}0.\ (\forall \tau{\in}\{0..t\}.\ G\ (s + \tau *_R c)) \longrightarrow Q\ (s + t *_R c))$
shows **{***P***}** $(x' = (\lambda s.\ c)\ \&\ G)$ **{***Q***}**
⟨*proof*⟩

lemma *diff-weak-rule*:
assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows **{***P***}** $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$ **{***Q***}**
⟨*proof*⟩

lemma *diff-cut-rule*:
assumes *wp-C*:*rel-kat.Hoare* $\lceil P \rceil\ (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)\ \lceil C \rceil$
and *wp-Q*:*rel-kat.Hoare* $\lceil P \rceil\ (x' = f\ \&\ (\lambda\ s.\ G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ \lceil Q \rceil$
shows **{***P***}** $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$ **{***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 *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 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* ⇒ *real^2* (‹*f*›)
where $f\ s \equiv (\chi\ i.\ \textit{if } i{=}1 \textit{ then } s\$2 \textit{ else } -s\$1)$

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

— Verified with annotated dynamics

lemma *pendulum-dyn*: $\{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}$ *EVOL* φ *G T* $\{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}$
⟨*proof*⟩

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

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

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

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

7.6.2 Bouncing Ball

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

abbreviation *fball* :: *real* ⇒ *real^2* ⇒ *real^2* (‹*f*›)
where *f g s* ≡ (*χ i. if i=1 then s$2 else g*)

abbreviation *ball-flow* :: *real* ⇒ *real* ⇒ *real^2* ⇒ *real^2* (‹*φ*›)
where *φ g τ s* ≡ (*χ i. if i=1 then g · τ ^ 2/2 + s$2 · τ + s$1 else g · τ + s$2*)

— Verified with differential invariants

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

lemma [*bb-real-arith*]:
assumes *0 > g* **and** *inv*: *2 · g · x − 2 · g · h = v · v*
shows (*x*::*real*) ≤ *h*
⟨*proof*⟩

lemma *fball-invariant*:
fixes *g h* :: *real*
defines *dinv*: *I* ≡ (*λs. 2 · g · s$1 − 2 · g · h − (s$2 · s$2) = 0*)
shows *diff-invariant I* (*λt. f g*) (*λs. UNIV*) *UNIV 0 G*
⟨*proof*⟩

lemma *bouncing-ball-inv*: *g < 0* ⟹ *h ≥ 0* ⟹
{*λs. s$1 = h ∧ s$2 = 0***}**
(*LOOP*
((*x´= f g* & (*λ s. s$1 ≥ 0*) *DINV* (*λs. 2 · g · s$1 − 2 · g · h − s$2 · s$2 = 0*));
(*IF* (*λ s. s$1 = 0*) *THEN* (*2 ::= (λs. − s$2)*) *ELSE skip*))
INV (*λs. 0 ≤ s$1 ∧ 2 · g · s$1 = 2 · g · h + s$2 · s$2*))
{*λs. 0 ≤ s$1 ∧ s$1 ≤ h***}**
⟨*proof*⟩

lemma [*bb-real-arith*]:
assumes *invar*: *2 · g · x = 2 · g · h + v · v*
and *pos*: *g · τ^2 / 2 + v · τ + (x::real) = 0*
shows *2 · g · h + (− (g · τ) − v) · (− (g · τ) − v) = 0*
and *2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0*
⟨*proof*⟩

lemma [*bb-real-arith*]:
assumes *invar*: *2 · g · x = 2 · g · h + v · v*
shows *2 · g · (g · τ^2 / 2 + v · τ + (x::real)) =*
2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *bouncing-ball-dyn*: *g < 0* ⟹ *h ≥ 0* ⟹
{*λs. s$1 = h ∧ s$2 = 0***}**
(*LOOP*
((*EVOL* (*φ g*) (*λ s. s$1 ≥ 0*) *T*);
(*IF* (*λ s. s$1 = 0*) *THEN* (*2 ::= (λs. − s$2)*) *ELSE skip*))

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

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

lemma *bouncing-ball-flow*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{\lambda s.\ s\$1 = h \wedge s\$2 = 0\}$
(*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*: $g < (0::real) \Longrightarrow 0 \leq h \Longrightarrow$
$2 ::= (\lambda s.\ -\ s\$2) \leq$ *rel-R*
$\lceil \lambda s.\ s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
⟨*proof*⟩

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows *rel-R* $\lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil$ $\lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil \geq$
(*LOOP*
((*EVOL* $(\varphi\ g)$ $(\lambda\ s.\ s\$1 \geq 0)$ *T*);
(*IF* $(\lambda\ s.\ s\$1 = 0)$ *THEN* $(2 ::= (\lambda s.\ -\ s\$2))$ *ELSE skip*))
INV $(\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
⟨*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 *0*, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -\ a * (T - U)$ where $U = L \geq 0$ when the heater is on, and $U = 0$ when it is off. We use *1* to denote the room's temperature, *2* is time as measured by the thermostat's chronometer, and *3* is a variable to save temperature measurements. Finally, *4* states whether the heater is on $(s\$4 = 1)$ or off $(s\$4 = 0)$. We prove that the thermostat keeps the room's temperature between *Tmin* and *Tmax*.

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

abbreviation *therm-guard* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (‹*G*›)
  where *G Tmin Tmax a L s* ≡ (*s$2* ≤ − (*ln* ((*L*−(*if L=0 then Tmin else Tmax*))/(*L*−*s$3*)))/*a*)

abbreviation *therm-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (‹*I*›)
  where *I Tmin Tmax s* ≡ *Tmin* ≤ *s$1* ∧ *s$1* ≤ *Tmax* ∧ (*s$4* = *0* ∨ *s$4* = *1*)

abbreviation *therm-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*φ*›)
  where *φ a L τ s* ≡ (*χ i. if i = 1 then − exp*(−*a* ∗ *τ*) ∗ (*L* − *s$1*) + *L else*
  (*if i = 2 then τ* + *s$2 else s$i*))

— Verified with the flow

lemma *norm-diff-therm-dyn*: *0* < *a* ⟹ ‖*f a L s*$_1$ − *f a L s*$_2$‖ = |*a*| ∗ |*s*$_1$*$1* − *s*$_2$*$1*|
⟨*proof*⟩

lemma *local-lipschitz-therm-dyn*:
  assumes *0* < (*a*::*real*)
  shows *local-lipschitz UNIV UNIV* (λ*t*::*real. f a L*)
  ⟨*proof*⟩

lemma *local-flow-therm*: *a* > *0* ⟹ *local-flow* (*f a L*) *UNIV UNIV* (*φ a L*)
  ⟨*proof*⟩

lemma *therm-dyn-down-real-arith*:
  assumes *a* > *0* **and** *Thyps*: *0* < *Tmin Tmin* ≤ *T T* ≤ *Tmax*
    and *thyps*: *0* ≤ (*τ*::*real*) ∀*τ*∈{*0*..*τ*}. *τ* ≤ − (*ln* (*Tmin* / *T*) / *a*)
  shows *Tmin* ≤ *exp* (−*a* ∗ *τ*) ∗ *T* **and** *exp* (−*a* ∗ *τ*) ∗ *T* ≤ *Tmax*
⟨*proof*⟩

lemma *therm-dyn-up-real-arith*:
  assumes *a* > *0* **and** *Thyps*: *Tmin* ≤ *T T* ≤ *Tmax Tmax* < (*L*::*real*)
    and *thyps*: *0* ≤ *τ* ∀*τ*∈{*0*..*τ*}. *τ* ≤ − (*ln* ((*L* − *Tmax*) / (*L* − *T*)) / *a*)
  shows *L* − *Tmax* ≤ *exp* (−(*a* ∗ *τ*)) ∗ (*L* − *T*)
    and *L* − *exp* (−(*a* ∗ *τ*)) ∗ (*L* − *T*) ≤ *Tmax*
    and *Tmin* ≤ *L* − *exp* (−(*a* ∗ *τ*)) ∗ (*L* − *T*)
⟨*proof*⟩

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* ≤ *τ* **and** *0* < *Tmin* **and** *Tmax* < *L*
  shows {*I Tmin Tmax*}
  (*LOOP* (
    — control
    (*2* ::= (λ*s. 0*));
    (*3* ::= (λ*s. s$1*));

(*IF* (λ*s*. *s*$*4* = *0* ∧ *s*$*3* ≤ *Tmin* + *1*) *THEN*
(*4* ::= (λ*s*.*1*))
ELSE IF (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *Tmax* − *1*) *THEN*
(*4* ::= (λ*s*.*0*))
ELSE skip);
— dynamics
(*IF* (λ*s*. *s*$*4* = *0*) *THEN*
(*x´*= (λ*t*. *f a 0*) & *G Tmin Tmax a 0 on* (λ*s*. {*0*..τ}) *UNIV* @ *0*)
ELSE
(*x´*= (λ*t*. *f a L*) & *G Tmin Tmax a L on* (λ*s*. {*0*..τ}) *UNIV* @ *0*))
) *INV I Tmin Tmax*)
{*I Tmin Tmax***}**
⟨*proof*⟩

lemma *R-therm-dyn-down*:
assumes *a* > *0* **and** *0* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s*. *s*$*4* = *0* ∧ *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x´*= (λ*t*. *f a 0*) & *G Tmin Tmax a 0 on* (λ*s*. {*0*..τ}) *UNIV* @ *0*)
⟨*proof*⟩

lemma *R-therm-dyn-up*:
assumes *a* > *0* **and** *0* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s*. *s*$*4* ≠ *0* ∧ *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*x´*= (λ*t*. *f a L*) & *G Tmin Tmax a L on* (λ*s*. {*0*..τ}) *UNIV* @ *0*)
⟨*proof*⟩

lemma *R-therm-dyn*:
assumes *a* > *0* **and** *0* ≤ τ **and** *0* < *Tmin* **and** *Tmax* < *L*
shows *rel-R* ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ⌈*I Tmin Tmax*⌉ ≥
(*IF* (λ*s*. *s*$*4* = *0*) *THEN*
(*x´*= (λ*t*. *f a 0*) & *G Tmin Tmax a 0 on* (λ*s*. {*0*..τ}) *UNIV* @ *0*)
ELSE
(*x´*= (λ*t*. *f a L*) & *G Tmin Tmax a L on* (λ*s*. {*0*..τ}) *UNIV* @ *0*))
⟨*proof*⟩

lemma *R-therm-assign1*: *rel-R* ⌈*I Tmin Tmax*⌉ ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0*⌉ ≥ (*2* ::= (λ*s*. *0*))
⟨*proof*⟩

lemma *R-therm-assign2*:
rel-R ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0*⌉ ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ≥ (*3* ::= (λ*s*. *s*$*1*))
⟨*proof*⟩

lemma *R-therm-ctrl*:
rel-R ⌈*I Tmin Tmax*⌉ ⌈λ*s*. *I Tmin Tmax s* ∧ *s*$*2* = *0* ∧ *s*$*3* = *s*$*1*⌉ ≥
(*2* ::= (λ*s*. *0*));

```
  (3 ::= (λs. s$1));
  (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
    (4 ::= (λs.1))
   ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax − 1) THEN
    (4 ::= (λs.0))
   ELSE skip)
  ⟨proof⟩

lemma R-therm-loop: rel-R ⌈I Tmin Tmax⌉ ⌈I Tmin Tmax⌉ ≥
  (LOOP
    rel-R ⌈I Tmin Tmax⌉ ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉;
    rel-R ⌈λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1⌉ ⌈I Tmin Tmax⌉
  INV I Tmin Tmax)
  ⟨proof⟩

lemma R-thermostat-flow:
  assumes a > 0 and 0 ≤ τ and 0 < Tmin and Tmax < L
  shows rel-R ⌈I Tmin Tmax⌉ ⌈I Tmin Tmax⌉ ≥
  (LOOP (
    — control
    (2 ::= (λs. 0));(3 ::= (λs. s$1));
    (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
      (4 ::= (λs.1))
     ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax − 1) THEN
      (4 ::= (λs.0))
     ELSE skip);
    — dynamics
    (IF (λs. s$4 = 0) THEN
      (x´= (λt. f a 0) & G Tmin Tmax a 0 on (λs. {0..τ}) UNIV @ 0)
    ELSE
      (x´= (λt. f a L) & G Tmin Tmax a L on (λs. {0..τ}) UNIV @ 0))
  ) 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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
  where $f\ k\ s \equiv (\chi\ i.\ \mathit{if}\ i = 2\ \mathit{then}\ 1\ \mathit{else}\ (\mathit{if}\ i = 1\ \mathit{then}\ k\ \mathit{else}\ 0))$

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

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

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

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

— Verified with the flow

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

lemma *tank-arith*:
  assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** $c_o < c_i$
  shows $\forall \tau{\in}\{0..\tau\}.\ \tau \leq -((hmin - y)\ /\ c_o) \Longrightarrow\ hmin \leq y - c_o * \tau$
    and $\forall \tau{\in}\{0..\tau\}.\ \tau \leq (hmax - y)\ /\ (c_i - c_o) \Longrightarrow\ (c_i - c_o) * \tau + y \leq hmax$
    and $hmin \leq y \Longrightarrow hmin \leq (c_i - c_o) \cdot \tau + y$
    and $y \leq hmax \Longrightarrow y - c_o \cdot \tau \leq hmax$
  ⟨*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* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
    (*IF* (λ*s. s$4 = 0* $\wedge$ *s$3* $\leq$ *hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
    (*IF* (λ*s. s$4 = 1* $\wedge$ *s$3* $\geq$ *hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
    — dynamics
     (*IF* (λ*s. s$4 = 0*) *THEN* (*x´*= ($\lambda$*t. f* ($c_i - c_o$)) & *G hmax* ($c_i - c_o$) *on* ($\lambda$*s.* $\{0..\tau\}$) *UNIV* @ *0*)
      ELSE (*x´*= (λ*t. f* ($-c_o$)) & *G hmin* ($-c_o$) *on* (λ*s.* $\{0..\tau\}$) *UNIV* @ *0*)))
  INV I hmin hmax)

{I *hmin hmax*}
⟨*proof*⟩

lemma *tank-diff-inv*:
$0 \leq \tau \Longrightarrow$ *diff-invariant* (*dI hmin hmax k*) (λ*t*. *f k*) (λ*s*. {*0*..τ}) *UNIV 0 Guard*
⟨*proof*⟩

lemma *tank-inv-arith1*:
assumes $0 \leq (\tau::real)$ **and** $c_o < c_i$ **and** *b*: $hmin \leq y_0$ **and** *g*: $\tau \leq (hmax - y_0) / (c_i - c_o)$
shows $hmin \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq hmax$
⟨*proof*⟩

lemma *tank-inv-arith2*:
assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** *b*: $y_0 \leq hmax$ **and** *g*: $\tau \leq -((hmin - y_0) / c_o)$
shows $hmin \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq hmax$
⟨*proof*⟩

lemma *tank-inv*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows **{**I *hmin hmax*}
(*LOOP*
— control
(*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0 ∧ s$3 ≤ hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1 ∧ s$3 ≥ hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s. s$4 = 0*) *THEN*
(*x´= (λt. f (*c_i*−*c_o*)) & G hmax (*c_i*−*c_o*) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (*c_i*−*c_o*)))*
ELSE
(*x´= (λt. f (−*c_o*)) & G hmin (−*c_o*) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (−*c_o*))))*)
INV I hmin hmax)
{I *hmin hmax*}
⟨*proof*⟩

abbreviation *tank-ctrl* :: *real* ⇒ *real* ⇒ (*real^4*) *rel* (‹*ctrl*›)
where *ctrl hmin hmax* ≡ ((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0 ∧ s$3 ≤ hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1 ∧ s$3 ≥ hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*)))

abbreviation *tank-dyn-dinv* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *rel* (‹*dyn*›)
where *dyn* c_i c_o *hmin hmax* τ ≡ (*IF* (λ*s. s$4 = 0*) *THEN*
(*x´= (λt. f (*c_i*−*c_o*)) & G hmax (*c_i*−*c_o*) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (*c_i*−*c_o*)))*
ELSE

(x´= (λt. f (−c_o)) & G hmin (−c_o) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (−c_o))))

abbreviation *tank-dinv* c_i c_o *hmin hmax* τ ≡
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*⌉ ≥
(*LOOP*
— control
((2 ::=(λs.0));(3 ::=(λs. s$1));
(*IF* (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) *THEN* (4 ::= (λs.1)) *ELSE*
(*IF* (λs. s$4 = 1 ∧ s$3 ≥ hmax − 1) *THEN* (4 ::= (λs.0)) *ELSE skip*));
— dynamics
(*IF* (λs. s$4 = 0) *THEN*
(x´= (λt. f (c_i−c_o)) & G hmax (c_i−c_o) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (c_i−c_o)))
ELSE
(x´= (λt. f (−c_o)) & G hmin (−c_o) on (λs. {0..τ}) 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 $n\ f = (\lambda x.\ \mathit{if}\ ((f_\bullet)\ x = \{\})\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})^\bullet$

lemma *nd-fun-n-op-one*[*nd-fun-ka*]: $n\ (n\ (1::'a\ \mathit{nd\text{-}fun})) = 1$
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** $x::'a\ \mathit{nd\text{-}fun}$
⟨*proof*⟩

instance
⟨*proof*⟩

end

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

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

instance
⟨*proof*⟩

end

7.7.1 Regular programs

Lemmas for manipulation of predicates in the relational model

type-synonym $'a\ \mathit{pred} = 'a \Rightarrow \mathit{bool}$

unbundle *no floor-ceiling-syntax*
no-notation *tau* (‹τ›)
and *Relation.relcomp* (**infixl** ‹;› 75)
and *proto-near-quantale-class.bres* (**infixr** ‹→› 60)

definition $\mathit{p2ndf} :: 'a\ \mathit{pred} \Rightarrow 'a\ \mathit{nd\text{-}fun}$ (‹(1⌈-⌉)›)
where $\lceil Q \rceil \equiv (\lambda\ x::'a.\ \{s::'a.\ s = x \wedge Q\ s\})^\bullet$

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

Lemmas for verification condition generation

abbreviation *ndfunHoare* (‹**{-}-{-}**›)

where $\{P\}X\{Q\} \equiv \mathit{Hoare}\ \lceil P\rceil\ X\ \lceil Q\rceil$

lemma *ndfun-kat-H*: $\{P\}\ X\ \{Q\} \longleftrightarrow (\forall\, s\ s'.\ P\ s \longrightarrow s' \in (X_\bullet)\ s \longrightarrow Q\ s')$
⟨*proof*⟩

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

lemma *sH-skip*[*simp*]: $\{P\}\ \mathit{skip}\ \{Q\} \longleftrightarrow \lceil P\rceil \leq \lceil Q\rceil$
⟨*proof*⟩

lemma *H-skip*: $\{P\}\ \mathit{skip}\ \{P\}$
⟨*proof*⟩

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

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

lemma *sH-abort*[*simp*]: $\{P\}\ \mathit{abort}\ \{Q\} \longleftrightarrow \mathit{True}$
⟨*proof*⟩

lemma *H-abort*: $\{P\}\ \mathit{abort}\ \{Q\}$
⟨*proof*⟩

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

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

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

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

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 $\mathit{seq\text{-}seq} :: 'a\ \mathit{nd\text{-}fun} \Rightarrow 'a\ \mathit{nd\text{-}fun} \Rightarrow 'a\ \mathit{nd\text{-}fun}$ (**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 $\forall s.\ P\ s \longrightarrow K\ (vec\text{-}lambda\ (((\$)\ s)(x := e\ s)))$
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* ⇒ *'a nd-fun* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (‹*IF - THEN - ELSE -*› [*64*,*64*] *63*)
where *IF B THEN X ELSE Y* ≡ *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* ⇒ *'a pred* ⇒ *'a nd-fun* ⇒ *'a nd-fun* (‹*WHILE - INV - DO -*› [*64*,*64*,*64*] *63*)
where *WHILE B INV I DO X* ≡ *kat-while-inv* ⌈*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*) $\{\lambda s.\ P\ s \wedge \neg\ B\ s\}$
⟨*proof*⟩

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

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

lemma *H-loopI*: $\{I\}\ X\ \{I\} \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow \{P\}\ (LOOP\ X\ INV\ I)\ \{Q\}$
⟨*proof*⟩

7.7.2 Evolution commands

definition *g-evol* :: $(('a::ord) \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b\ pred \Rightarrow ('b \Rightarrow 'a\ set) \Rightarrow 'b\ nd\text{-}fun$ (‹*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 $\{P\}\ (EVOL\ \varphi\ G\ U)\ \{Q\} = (\forall s.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

lemma *H-g-evol*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
assumes $P = (\lambda s.\ (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows $\{P\}\ (EVOL\ \varphi\ G\ U)\ \{Q\}$
⟨*proof*⟩

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

lemma *H-g-orbital*:
$P = (\lambda s.\ (\forall X{\in}ivp\text{-}sols\ f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \Longrightarrow$
$\{P\}\ (x´= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{Q\}$
⟨*proof*⟩

lemma *sH-g-orbital*: $\{P\}\ (x´= f\ \&\ G\ on\ U\ S\ @\ t_0)\ \{Q\} =$
$(\forall s.\ P\ s \longrightarrow (\forall X{\in}ivp\text{-}sols\ f\ U\ S\ t_0\ s.\ \forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)))$
⟨*proof*⟩

context *local-flow*
begin

lemma *sH-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
shows $\{P\}\ (x´= (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \{Q\} =$
$(\forall s{\in}S.\ P\ s \longrightarrow (\forall t{\in}U\ s.\ (\forall \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

lemma *H-g-ode-subset*:

assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ s \in S \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
shows $\{P\}\ (x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0)\ \{Q\}$
⟨*proof*⟩

lemma *sH-g-ode*: $\{P\}\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ T)\ S\ @\ 0)\ \{Q\} =$
$(\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*proof*⟩

lemma *sH-g-ode-ivl*: $t \geq 0 \Longrightarrow t \in T \Longrightarrow \{P\}\ (x' = (\lambda t.\ f)\ \&\ G\ on\ (\lambda s.\ \{0..t\})\ S\ @\ 0)\ \{Q\} =$
$(\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}\{0..t\}.\ (\forall\, \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
⟨*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* :: $(real \Rightarrow ('a{::}banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow ('a \Rightarrow real\ set) \Rightarrow 'a\ set \Rightarrow$
$real \Rightarrow 'a\ pred \Rightarrow 'a\ nd\text{-}fun$ (‹(1x´=- & - on - - @ - DINV -)›)
where $(x' = f\ \&\ G\ on\ U\ S\ @\ t_0\ DINV\ I) = (x' = f\ \&\ G\ on\ U\ S\ @\ t_0)$

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

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

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

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

7.8 Refinement Components

lemma *R-skip*: $(\forall\, s.\ P\ s \longrightarrow Q\ s) \Longrightarrow 1 \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$

⟨*proof*⟩

lemma *R-abort*: $abort \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-seq*: $(Ref\ \lceil P \rceil\ \lceil R \rceil)\ ;\ (Ref\ \lceil R \rceil\ \lceil Q \rceil) \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-seq-law*: $X \leq Ref\ \lceil P \rceil\ \lceil R \rceil \Longrightarrow Y \leq Ref\ \lceil R \rceil\ \lceil Q \rceil \Longrightarrow X;\ Y \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemmas *R-seq-mono* = *mult-isol-var*

— Nondeterministic Choice

lemma *R-choice*: $(Ref\ \lceil P \rceil\ \lceil Q \rceil) + (Ref\ \lceil P \rceil\ \lceil Q \rceil) \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-choice-law*: $X \leq Ref\ \lceil P \rceil\ \lceil Q \rceil \Longrightarrow Y \leq Ref\ \lceil P \rceil\ \lceil Q \rceil \Longrightarrow X + Y \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-choice-mono*: $P' \leq P \Longrightarrow Q' \leq Q \Longrightarrow P' + Q' \subseteq P + Q$
⟨*proof*⟩

lemma *R-assign*: $(x ::= e) \leq Ref\ \lceil \lambda s.\ P\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j) \rceil\ \lceil P \rceil$
⟨*proof*⟩

lemma *R-assign-law*:
$(\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j)) \Longrightarrow (x ::= e) \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-assignl*:
$P = (\lambda s.\ R\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow (x ::= e)\ ;\ Ref\ \lceil R \rceil\ \lceil Q \rceil \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-assignr*:
$R = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \Longrightarrow Ref\ \lceil P \rceil\ \lceil R \rceil;\ (x ::= e) \leq Ref\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma $(x ::= e)\ ;\ Ref\ \lceil Q \rceil\ \lceil Q \rceil \leq Ref\ \lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma $Ref\ \lceil Q \rceil\ \lceil (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := e\ s))\ j)) \rceil;\ (x ::= e) \leq Ref\ \lceil Q \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-nondet-assign*: $(x ::= ?) \leq Ref\ \lceil\lambda s.\ \forall k.\ P\ (\chi\ j.\ (((\$)\ s)(x := k))\ j)\rceil\ \lceil P\rceil$
⟨*proof*⟩

lemma *R-nondet-assign-law*:
$(\forall s.\ P\ s \longrightarrow (\forall k.\ Q\ (\chi\ j.\ (((\$)\ s)(x := k))\ j))) \Longrightarrow (x ::= ?) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-cond*:
$(IF\ B\ THEN\ Ref\ \lceil\lambda s.\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil\ ELSE\ Ref\ \lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-cond-mono*: $X \leq X' \Longrightarrow Y \leq Y' \Longrightarrow (IF\ P\ THEN\ X\ ELSE\ Y) \leq IF\ P\ THEN\ X'\ ELSE\ Y'$
⟨*proof*⟩

lemma *R-cond-law*: $X \leq Ref\ \lceil\lambda s.\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil \Longrightarrow Y \leq Ref\ \lceil\lambda s.\ \neg\ B\ s \wedge P\ s\rceil\ \lceil Q\rceil \Longrightarrow$
$(IF\ B\ THEN\ X\ ELSE\ Y) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-while*: $K = (\lambda s.\ P\ s \wedge \neg\ B\ s) \Longrightarrow$
$WHILE\ B\ INV\ I\ DO\ (Ref\ \lceil\lambda s.\ P\ s \wedge B\ s\rceil\ \lceil P\rceil) \leq Ref\ \lceil P\rceil\ \lceil K\rceil$
⟨*proof*⟩

lemma *R-whileI*:
$X \leq Ref\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil\lambda s.\ I\ s \wedge B\ s\rceil \Longrightarrow \lceil\lambda s.\ I\ s \wedge \neg\ B\ s\rceil \leq \lceil Q\rceil \Longrightarrow$
$WHILE\ B\ INV\ I\ DO\ X \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

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

lemma *R-while-law*: $X \leq Ref\ \lceil\lambda s.\ P\ s \wedge B\ s\rceil\ \lceil P\rceil \Longrightarrow Q = (\lambda s.\ P\ s \wedge \neg\ B\ s) \Longrightarrow$
$(WHILE\ B\ INV\ I\ DO\ X) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-loop*: $X \leq Ref\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow LOOP\ X\ INV\ I \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-loopI*: $X \leq Ref\ \lceil I\rceil\ \lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I\rceil \leq \lceil Q\rceil \Longrightarrow LOOP\ X\ INV\ I \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma *R-loop-mono*: $X \leq X' \Longrightarrow LOOP\ X\ INV\ I \leq LOOP\ X'\ INV\ I$

⟨*proof*⟩

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

lemma *R-g-evol-law*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall\, s.\ P\ s \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow (EVOL\ \varphi\ G\ U) \leq Ref\ \lceil P\rceil\ \lceil Q\rceil$
⟨*proof*⟩

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

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

lemma
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $EVOL\ \varphi\ G\ U\ ;\ Ref\ \lceil Q\rceil\ \lceil Q\rceil \leq Ref\ \lceil \lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil\ \lceil Q\rceil$
⟨*proof*⟩

lemma
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $Ref\ \lceil Q\rceil\ \lceil \lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)\rceil;\ EVOL\ \varphi\ G\ U \leq Ref\ \lceil Q\rceil\ \lceil Q\rceil$
⟨*proof*⟩

context *local-flow*
begin

lemma *R-g-ode-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge is\text{-}interval\ (U\ s) \wedge U\ s \subseteq T$
shows $(x' = (\lambda t.\ f)\ \&\ G\ on\ U\ S\ @\ 0) \leq Ref\ \lceil \lambda s.\ s{\in}S \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}down\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))\rceil\ \lceil P\rceil$
⟨*proof*⟩

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

assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
shows $(\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
$\Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ U\ S\ @\ 0) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-odel-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $P = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s))$
shows $(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ U\ S\ @\ 0)\ ;\ \mathit{Ref}\ \lceil R \rceil\ \lceil Q \rceil \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-oder-subset*:
assumes $\bigwedge s.\ s \in S \Longrightarrow 0 \in U\ s \wedge$ *is-interval* $(U\ s) \wedge U\ s \subseteq T$
and $R = (\lambda s.\ \forall\, t{\in}U\ s.\ (\forall\, \tau{\in}\mathit{down}\ (U\ s)\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\mathit{Ref}\ \lceil P \rceil\ \lceil R \rceil;\ (x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ U\ S\ @\ 0) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-ode*: $(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq$
$\mathit{Ref}\ \lceil \lambda s.\ s{\in}S \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}\mathit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)) \rceil\ \lceil P \rceil$
⟨*proof*⟩

lemma *R-g-ode-law*: $(\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}T.\ (\forall\, \tau{\in}\mathit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-odel*: $P = (\lambda s.\ \forall\, t{\in}T.\ (\forall\, \tau{\in}\mathit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ T)\ S\ @\ 0)\ ;\ \mathit{Ref}\ \lceil R \rceil\ \lceil Q \rceil \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-oder*: $R = (\lambda s.\ \forall\, t{\in}T.\ (\forall\, \tau{\in}\mathit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \Longrightarrow$
$\mathit{Ref}\ \lceil P \rceil\ \lceil R \rceil;\ (x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ T)\ S\ @\ 0) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

lemma *R-g-ode-ivl*:
$t \geq 0 \Longrightarrow t \in T \Longrightarrow (\forall\, s{\in}S.\ P\ s \longrightarrow (\forall\, t{\in}\{0..t\}.\ (\forall\, \tau{\in}\{0..t\}.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \Longrightarrow$
$(x' = (\lambda t.\ f)\ \&\ G\ \mathit{on}\ (\lambda s.\ \{0..t\})\ S\ @\ 0) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant I f T S* t_0 $G \Longrightarrow \lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil \lambda s.\ I\ s \wedge G\ s \rceil \leq \lceil Q \rceil \Longrightarrow$
$(x' = f\ \&\ G\ \mathit{on}\ T\ S\ @\ t_0\ \mathit{DINV}\ I) \leq \mathit{Ref}\ \lceil P \rceil\ \lceil Q \rceil$
⟨*proof*⟩

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*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a nd-fun* (‹(*1x´=- & -*)›)

where (*x´=f & G*) ≡ (*x´= (λt. f) & G on (λs. {t. t ≥ 0}) UNIV @ 0*)

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

lemma *diff-solve-rule1*:
assumes *local-flow f UNIV UNIV φ*
and ∀ *s. P s* ⟶ (∀ *t*≥*0*. (∀ *τ*∈{*0..t*}. *G* (*φ τ s*)) ⟶ *Q* (*φ t s*))
shows **{***P***}** (*x´= f & G*) **{***Q***}**
⟨*proof*⟩

lemma *diff-solve-rule2*:
fixes *c*::'*a*::{*heine-borel*, *banach*}
assumes ∀ *s. P s* ⟶ (∀ *t*≥*0*. (∀ *τ*∈{*0..t*}. *G* (*s* + *τ* ∗$_R$ *c*)) ⟶ *Q* (*s* + *t* ∗$_R$ *c*))
shows **{***P***}** (*x´=*(*λs. c*) *& G*) **{***Q***}**
⟨*proof*⟩

lemma *diff-weak-rule*:
assumes ⌈*G*⌉ ≤ ⌈*Q*⌉
shows **{***P***}** (*x´= f & G on U S @ t*$_0$) **{***Q***}**
⟨*proof*⟩

lemma *diff-cut-rule*:
assumes *wp-C*:*Hoare* ⌈*P*⌉ (*x´= f & G on U S @ t*$_0$) ⌈*C*⌉
and *wp-Q*:*Hoare* ⌈*P*⌉ (*x´= f & (λ s. G s ∧ C s) on U S @ t*$_0$) ⌈*Q*⌉
shows **{***P***}** (*x´= f & G on U S @ t*$_0$) **{***Q***}**
⟨*proof*⟩

lemma *diff-inv-rule*:
assumes ⌈*P*⌉ ≤ ⌈*I*⌉ **and** *diff-invariant I f U S t*$_0$ *G* **and** ⌈*I*⌉ ≤ ⌈*Q*⌉
shows **{***P***}** (*x´= f & G on U S @ t*$_0$) **{***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* ⇒ *real^2* (‹*f*›)
where *f s* ≡ (*χ i. if i=1 then s$2 else −s$1*)

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

— Verified with annotated dynamics

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

lemma *pendulum-inv*: $\{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}\ (x´=f\ \&\ G)\ \{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}$
⟨*proof*⟩

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

lemma *pendulum-flow*: $\{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}\ (x´=f\ \&\ G)\ \{\lambda s.\ r^2 = (s\ \$\ 1)^2 + (s\ \$\ 2)^2\}$
⟨*proof*⟩

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

7.10.2 Bouncing Ball

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

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

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

— Verified with differential invariants

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

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

lemma *fball-invariant*:
fixes $g\ h :: real$
defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows *diff-invariant* $I\ (\lambda t.\ f\ g)\ (\lambda s.\ UNIV)\ UNIV\ 0\ G$
⟨*proof*⟩

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{\lambda s.\ s\$1 = h \wedge s\$2 = 0\}$
$(LOOP$
$((x' = f\ g\ \&\ (\lambda\ s.\ s\$1 \geq 0)\ DINV\ (\lambda s.\ 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2 = 0));$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
$)\ \{\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
⟨*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*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow$
$\{\lambda s.\ s\$1 = h \wedge s\$2 = 0\}$
$(LOOP$
$((EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
$)\ \{\lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h\}$
⟨*proof*⟩

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

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

⟨*proof*⟩

lemma *R-bb-assign*: $g < (0::real) \Longrightarrow 0 \leq h \Longrightarrow$
$2 ::= (\lambda s.\ -\ s\$2) \leq Ref$
$\lceil \lambda s.\ s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
$\lceil \lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$
⟨*proof*⟩

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows $Ref\ \lceil \lambda s.\ s\$1 = h \wedge s\$2 = 0 \rceil\ \lceil \lambda s.\ 0 \leq s\$1 \wedge s\$1 \leq h \rceil \geq$
$(LOOP$
$((EVOL\ (\varphi\ g)\ (\lambda\ s.\ s\$1 \geq 0)\ T);$
$(IF\ (\lambda\ s.\ s\$1 = 0)\ THEN\ (2 ::= (\lambda s.\ -\ s\$2))\ ELSE\ skip))$
$INV\ (\lambda s.\ 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
⟨*proof*⟩

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* ⇒ *real* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f a L s* ≡ (χ *i*. *if i = 2 then 1 else* (*if i = 1 then* − *a* ∗ (*s$1* − *L*) *else 0*))

abbreviation *therm-guard* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (‹*G*›)
where *G Tmin Tmax a L s* ≡ (*s$2* ≤ − (*ln* ((*L*−(*if L=0 then Tmin else Tmax*))/(*L*−*s$3*)))/*a*)

abbreviation *therm-loop-inv* :: *real* ⇒ *real* ⇒ *real^4* ⇒ *bool* (‹*I*›)
where *I Tmin Tmax s* ≡ *Tmin* ≤ *s$1* ∧ *s$1* ≤ *Tmax* ∧ (*s$4* = *0* ∨ *s$4* = *1*)

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

— Verified with the flow

lemma *norm-diff-therm-dyn*: $0 < a \Longrightarrow \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$
⟨*proof*⟩

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

lemma *local-flow-therm*: $a > 0 \Longrightarrow$ *local-flow* $(f\ a\ L)$ *UNIV UNIV* $(\varphi\ a\ L)$
⟨*proof*⟩

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

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

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 $Ref\ \lceil \lambda s.\ s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil\ \lceil I\ Tmin\ Tmax \rceil \geq$
$(x' = (\lambda t.\ f\ a\ 0)\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0)$
⟨*proof*⟩

lemma *R-therm-dyn-up*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $Ref\ \lceil \lambda s.\ s\$4 \neq 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil\ \lceil I\ Tmin\ Tmax \rceil \geq$
$(x' = (\lambda t.\ f\ a\ L)\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0)$
⟨*proof*⟩

lemma *R-therm-dyn*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $Ref\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil\ \lceil I\ Tmin\ Tmax \rceil \geq$
$(IF\ (\lambda s.\ s\$4 = 0)\ THEN$
$(x' = (\lambda t.\ f\ a\ 0)\ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0)$
$ELSE$
$(x' = (\lambda t.\ f\ a\ L)\ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s.\ \{0..\tau\})\ UNIV\ @\ 0))$
⟨*proof*⟩

lemma *R-therm-assign1*: $Ref\ \lceil I\ Tmin\ Tmax \rceil\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \rceil \geq$
$(2 ::= (\lambda s.\ 0))$
⟨*proof*⟩

lemma *R-therm-assign2*:
$Ref\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \rceil\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil \geq (3 ::= (\lambda s.\ s\$1))$
⟨*proof*⟩

lemma *R-therm-ctrl*:
$Ref\ \lceil I\ Tmin\ Tmax \rceil\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil \geq$
$(2 ::= (\lambda s.\ 0));$
$(3 ::= (\lambda s.\ s\$1));$
$(IF\ (\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1)\ THEN$
$(4 ::= (\lambda s.1))$
$ELSE\ IF\ (\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1)\ THEN$
$(4 ::= (\lambda s.0))$
$ELSE\ skip)$
⟨*proof*⟩

lemma *R-therm-loop*: $Ref\ \lceil I\ Tmin\ Tmax \rceil\ \lceil I\ Tmin\ Tmax \rceil \geq$
$(LOOP$
$Ref\ \lceil I\ Tmin\ Tmax \rceil\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil;$
$Ref\ \lceil \lambda s.\ I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1 \rceil\ \lceil I\ Tmin\ Tmax \rceil$
$INV\ I\ Tmin\ Tmax)$
⟨*proof*⟩

lemma *R-thermostat-flow*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows *Ref* $\lceil I\ Tmin\ Tmax \rceil$ $\lceil I\ Tmin\ Tmax \rceil \geq$
(*LOOP* (
— control
(*2* ::= ($\lambda s.\ 0$));(*3* ::= ($\lambda s.\ s\$1$));
(*IF* ($\lambda s.\ s\$4 = 0 \wedge s\$3 \leq Tmin + 1$) *THEN*
(*4* ::= ($\lambda s.1$))
ELSE IF ($\lambda s.\ s\$4 = 1 \wedge s\$3 \geq Tmax - 1$) *THEN*
(*4* ::= ($\lambda s.0$))
ELSE skip);
— dynamics
(*IF* ($\lambda s.\ s\$4 = 0$) *THEN*
($x´ = (\lambda t.\ f\ a\ 0)$ & *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*›)

7.10.4 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* ⇒ *real^4* ⇒ *real^4* (‹*f*›)
where *f k s* ≡ (χ *i. if i = 2 then 1 else* (*if i = 1 then k else 0*))

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

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

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

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

— Verified with the flow

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

lemma *tank-arith*:
assumes *0* ≤ (*τ*::*real*) **and** *0* < c_o **and** c_o < c_i
shows ∀*τ*∈{*0*..*τ*}. *τ* ≤ − ((*hmin* − *y*) / c_o) ⟹ *hmin* ≤ *y* − c_o ∗ *τ*
and ∀*τ*∈{*0*..*τ*}. *τ* ≤ (*hmax* − *y*) / (c_i − c_o) ⟹ (c_i − c_o) ∗ *τ* + *y* ≤ *hmax*
and *hmin* ≤ *y* ⟹ *hmin* ≤ (c_i − c_o) · *τ* + *y*
and *y* ≤ *hmax* ⟹ *y* − c_o · *τ* ≤ *hmax*
⟨*proof*⟩

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

lemma *tank-flow*:
assumes *0* ≤ *τ* **and** *0* < c_o **and** c_o < c_i
shows **{***I hmin hmax***}**
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4* = *0* ∧ *s$3* ≤ *hmin* + *1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4* = *1* ∧ *s$3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s. s$4* = *0*) *THEN* (*x´*= (λ*t. f* (c_i−c_o)) & *G hmax* (c_i−c_o) *on* (λ*s.* {*0*..*τ*}) *UNIV* @ *0*)
ELSE (*x´*= (λ*t. f* (−c_o)) & *G hmin* (−c_o) *on* (λ*s.* {*0*..*τ*}) *UNIV* @ *0*)))
INV I hmin hmax)
{*I hmin hmax***}**
⟨*proof*⟩

lemma *tank-diff-inv*:
0 ≤ *τ* ⟹ *diff-invariant* (*dI hmin hmax k*) (λ*t. f k*) (λ*s.* {*0*..*τ*}) *UNIV 0 Guard*
⟨*proof*⟩

lemma *tank-inv-arith1*:
assumes *0* ≤ (*τ*::*real*) **and** c_o < c_i **and** *b*: *hmin* ≤ y_0 **and** *g*: *τ* ≤ (*hmax* − y_0) / (c_i − c_o)
shows *hmin* ≤ (c_i − c_o) · *τ* + y_0 **and** (c_i − c_o) · *τ* + y_0 ≤ *hmax*
⟨*proof*⟩

lemma *tank-inv-arith2*:
assumes $0 \leq (\tau::real)$ **and** $0 < c_o$ **and** b: $y_0 \leq hmax$ **and** g: $\tau \leq -((hmin - y_0) / c_o)$
shows $hmin \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq hmax$
⟨*proof*⟩

lemma *tank-inv*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows **{***I hmin hmax***}**
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0 ∧ s$3 ≤ hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1 ∧ s$3 ≥ hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s. s$4 = 0*) *THEN*
(*x´*= (λ*t. f* (*c_i*−*c_o*)) & *G hmax* (*c_i*−*c_o*) *on* (λ*s.* {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (*c_i*−*c_o*)))
ELSE
(*x´*= (λ*t. f* (−*c_o*)) & *G hmin* (−*c_o*) *on* (λ*s.* {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (−*c_o*)))))
INV I hmin hmax)
{*I hmin hmax***}**
⟨*proof*⟩

abbreviation *tank-ctrl* :: *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (‹*ctrl*›)
where *ctrl hmin hmax* ≡ ((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0 ∧ s$3 ≤ hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*
(*IF* (λ*s. s$4 = 1 ∧ s$3 ≥ hmax − 1*) *THEN* (*4* ::= (λ*s.0*)) *ELSE skip*)))

abbreviation *tank-dyn-dinv* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (‹*dyn*›)
where *dyn c_i c_o hmin hmax τ* ≡ (*IF* (λ*s. s$4 = 0*) *THEN*
(*x´*= (λ*t. f* (*c_i*−*c_o*)) & *G hmax* (*c_i*−*c_o*) *on* (λ*s.* {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (*c_i*−*c_o*)))
ELSE
(*x´*= (λ*t. f* (−*c_o*)) & *G hmin* (−*c_o*) *on* (λ*s.* {*0..τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (−*c_o*))))

abbreviation *tank-dinv c_i c_o hmin hmax τ* ≡ *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 *Ref* ⌈*I hmin hmax*⌉ ⌈*I hmin hmax*⌉ ≥
(*LOOP*
— control
((*2* ::=(λ*s.0*));(*3* ::=(λ*s. s$1*));
(*IF* (λ*s. s$4 = 0 ∧ s$3 ≤ hmin + 1*) *THEN* (*4* ::= (λ*s.1*)) *ELSE*

(*IF* (λ*s*. *s*$*4* = *1* ∧ *s*$*3* ≥ *hmax* − *1*) *THEN* (*4* ::= (λ*s*.*0*)) *ELSE skip*));
— dynamics
(*IF* (λ*s*. *s*$*4* = *0*) *THEN*
(*x´*= (λ*t*. *f* (*c*$_i$−*c*$_o$)) & *G hmax* (*c*$_i$−*c*$_o$) *on* (λ*s*. {*0*..*τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (*c*$_i$−*c*$_o$)))
ELSE
(*x´*= (λ*t*. *f* (−*c*$_o$)) & *G hmin* (−*c*$_o$) *on* (λ*s*. {*0*..*τ*}) *UNIV* @ *0 DINV* (*dI hmin hmax* (−*c*$_o$)))))
INV I hmin hmax)
⟨*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] R. Bohrer, V. Rahli, I. Vukotic, M. Völp, 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.