Hybrid KAT and rKAT

Jonathan Julián Huerta y Munive

25 octobre 2019

Table des matières

1 Verification components with KAT

In this section we derive the rules of Hoare Logic and a refinement calculus in KAT.

theory *KAT-rKAT-Prelims*
imports
KAT-and-DRA.PHL-KAT
Transformer-Semantics.Kleisli-Quantale
UTP.utp-pred-laws
UTP.utp-lift-parser
UTP.utp-lift-pretty
begin recall-syntax

purge-notation *Lattices.inf* (**infixl** $\sqcup$ *70*)
notation *Lattices.inf* (**infixl** $\sqcap$ *70*)
purge-notation *Lattices.sup* (**infixl** $\sqcap$ *65*)
notation *Lattices.sup* (**infixl** $\sqcup$ *65*)

1.1 Hoare logic derivation

no-notation *if-then-else* (*if - then - else - fi* [*64*,*64*,*64*] *63*)
and *while* (*while - do - od* [*64*,*64*] *63*)

context *kat*
begin

— Definitions of Hoare Triple

definition *Hoare* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (H) **where**
$H\ p\ x\ q \longleftrightarrow t\ p \cdot x \leq x \cdot t\ q$

lemma *H-consl*: $t\ p \leq t\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons1* **by** *blast*

lemma *H-consr*: $t\ q' \leq t\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$
using *Hoare-def phl-cons2* **by** *blast*

lemma *H-cons*: $t\ p \leq t\ p' \Longrightarrow t\ q' \leq t\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$
by (*simp add*: *H-consl H-consr*)

— Skip program

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

— Sequential composition

lemma *H-seq*: $H\ p\ x\ r \Longrightarrow H\ r\ y\ q \Longrightarrow H\ p\ (x \cdot y)\ q$
  by (*simp add*: *Hoare-def phl-seq*)

— Conditional statement

definition *ifthenelse* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (*if - then - else - fi* [*64*,*64*,*64*] *63*)
where
  if p then x else y fi $= (t\ p \cdot x + n\ p \cdot y)$

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

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

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

— While loop

definition *while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - od* [*64*,*64*] *63*) **where**
  while b do x od $= (t\ b \cdot x)^\star \cdot n\ b$

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

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

lemma *H-while*: $H\ (t\ p \cdot t\ r)\ x\ p \Longrightarrow H\ p$ (*while r do x od*) $(t\ p \cdot n\ r)$
proof −
  assume *a1*: $H\ (t\ p \cdot t\ r)\ x\ p$
  have $t\ (t\ p \cdot n\ r) = n\ r \cdot t\ p \cdot n\ r$
    using *n-preserve test-mult* **by** *presburger*

then show *?thesis*
using *a1 Hoare-def H-exp1 conway.phl.it-simr phl-export2 while-def* **by** *auto*
qed

lemma *H-while-inv*: $t\ p \leq t\ i \Longrightarrow t\ i \cdot n\ r \leq t\ q \Longrightarrow H\ (t\ i \cdot t\ r)\ x\ i \Longrightarrow H\ p$ $(while\ r\ inv\ i\ do\ x\ od)\ q$
by (*metis H-cons H-while test-mult while-inv-def*)

— Finite iteration

lemma *H-star*: $H\ i\ x\ i \Longrightarrow H\ i\ (x^\star)\ i$
unfolding *Hoare-def* **using** *star-sim2* **by** *blast*

lemma *H-star-inv*:
assumes $t\ p \leq t\ i$ **and** $H\ i\ x\ i$ **and** $(t\ i) \leq (t\ q)$
shows $H\ p\ (x^\star)\ q$
proof−
have $H\ i\ (x^\star)\ i$
using *assms*(*2*) *H-star* **by** *blast*
hence $H\ p\ (x^\star)\ i$
unfolding *Hoare-def* **using** *assms*(*1*) *phl-cons1* **by** *blast*
thus *?thesis*
unfolding *Hoare-def* **using** *assms*(*3*) *phl-cons2* **by** *blast*
qed

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

lemma *H-loop*: $H\ p\ x\ p \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ p$
unfolding *loopi-def* **by** (*rule H-star*)

lemma *H-loop-inv*: $t\ p \leq t\ i \Longrightarrow H\ i\ x\ i \Longrightarrow t\ i \leq t\ q \Longrightarrow H\ p\ (loop\ x\ inv\ i)\ q$
unfolding *loopi-def* **using** *H-star-inv* **by** *blast*

— Invariants

lemma *H-inv*: $t\ p \leq t\ i \Longrightarrow t\ i \leq t\ q \Longrightarrow H\ i\ x\ i \Longrightarrow H\ p\ x\ q$
by (*rule-tac* $p'=i$ **and** $q'=i$ **in** *H-cons*)

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

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

end

1.2 refinement KAT

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

begin

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

lemma *R2*: $H\ p\ x\ q \Longrightarrow x \leq Ref\ p\ q$
  by (*simp add*: *spec-def*)

lemma *R-cons*: $t\ p \leq t\ p' \Longrightarrow t\ q' \leq t\ q \Longrightarrow Ref\ p'\ q' \leq Ref\ p\ q$
proof –
  assume *h1*: $t\ p \leq t\ p'$ **and** *h2*: $t\ q' \leq t\ q$
  have $H\ p'\ (Ref\ p'\ q')\ q'$
    by (*simp add*: *R1*)
  hence $H\ p\ (Ref\ p'\ q')\ q$
    using *h1 h2 H-consl H-consr* **by** *blast*
  thus *?thesis*
    by (*rule R2*)
qed

— Abort and skip programs

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

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

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

— Sequential composition

lemma *R-seq*: (*Ref p r*) · (*Ref r q*) ≤ *Ref p q*
proof –
  have *H p* (*Ref p r*) *r* **and** *H r* (*Ref r q*) *q*
    by (*simp add*: *R1*)+
  hence *H p* ((*Ref p r*) · (*Ref r q*)) *q*
    by (*rule H-seq*)
  thus *?thesis*
    by (*rule R2*)
qed

— Conditional statement

lemma *R-cond*: *if v then* (*Ref* (*t v* · *t p*) *q*) *else* (*Ref* (*n v* · *t p*) *q*) *fi* ≤ *Ref p q*
proof –
  have *H* (*t v* · *t p*) (*Ref* (*t v* · *t p*) *q*) *q* **and** *H* (*n v* · *t p*) (*Ref* (*n v* · *t p*) *q*) *q*
    by (*simp add*: *R1*)+
  hence *H p* (*if v then* (*Ref* (*t v* · *t p*) *q*) *else* (*Ref* (*n v* · *t p*) *q*) *fi*) *q*
    by (*simp add*: *H-cond n-mult-comm*)
 thus *?thesis*
    by (*rule R2*)
qed

— While loop

lemma *R-while*: *while q do* (*Ref* (*t p* · *t q*) *p*) *od* ≤ *Ref p* (*t p* · *n q*)
proof –
  have *H* (*t p* · *t q*) (*Ref* (*t p* · *t q*) *p*) *p*
    by (*simp-all add*: *R1*)
  hence *H p* (*while q do* (*Ref* (*t p* · *t q*) *p*) *od*) (*t p* · *n q*)
    by (*simp add*: *H-while*)
  thus *?thesis*
    by (*rule R2*)
qed

— Finite iteration

lemma *R-star*: (*Ref i i*)* ≤ *Ref i i*
proof –
  have *H i* (*Ref i i*) *i*
    using *R1* **by** *blast*
  hence *H i* ((*Ref i i*)*) *i*
    using *H-star* **by** *blast*
  thus *Ref i i** ≤ *Ref i i*
    by (*rule R2*)
qed

lemma *R-loop*: *loop* (*Ref p p*) *inv i* $\leq$ *Ref p p*
  unfolding *loopi-def* **by** (*rule R-star*)

— Invariants

lemma *R-inv*: *t p* $\leq$ *t i* $\Longrightarrow$ *t i* $\leq$ *t q* $\Longrightarrow$ *Ref i i* $\leq$ *Ref p q*
  using *R-cons* **by** *force*

end

end

2 KAT Models

We show that relations and non-deterministic functions form Kleene algebras with tests.

theory *KAT-rKAT-Models*
  imports *KAT-rKAT-Prelims*

begin

2.1 Relational model

interpretation *rel-uq*: *unital-quantale Id* (*O*) $\bigcap$ $\bigcup$ ($\cap$) ($\subseteq$) ($\subset$) ($\cup$) {} *UNIV*
  by (*unfold-locales*, *auto*)

lemma *power-is-relpow*: *rel-uq.power X m* = *X* ^^ *m* **for** *X*::'*a rel*
proof (*induct m*)
  case *0* **show** *?case*
    by (*metis rel-uq.power-0 relpow.simps*(*1*))
  case *Suc* **thus** *?case*
    by (*metis rel-uq.power-Suc2 relpow.simps*(*2*))
qed

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

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

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

interpretation *rel-ka*: *kleene-algebra* ($\cup$) (*O*) *Id* {} ($\subseteq$) ($\subset$) *rtrancl*
proof
  fix *x y z* :: '*a rel*
  show *Id* $\cup$ *x O* x^* $\subseteq$ x^*
    by (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)

next
  fix $x\ y\ z$ :: $'a\ rel$
  assume $z \cup x\ O\ y \subseteq y$
  thus $x^*\ O\ z \subseteq y$
    by (*simp only*: *rel-star-contr*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductl*)
next
  fix $x\ y\ z$ :: $'a\ rel$
  assume $z \cup y\ O\ x \subseteq y$
  thus $z\ O\ x^* \subseteq y$
    by (*simp only*: *rel-star-contl*, *metis* (*lifting*) *SUP-le-iff rel-uq.power-inductr*)
qed

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

interpretation *rel-kat*: *kat* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl* $\lambda x.\ Id \cap (-\ x)$
  by (*unfold-locales*)

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

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

lemma *RdL-is-rRKAT*: $(\forall x.\ \{(x,x)\};\ R1 \subseteq \{(x,x)\};\ R2) = (R1 \subseteq R2)$
  by *auto*

2.2 State transformer model

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

definition *uexpr-nd-fun* :: $('a\ set,\ 'a)$ *uexpr* $\Rightarrow\ 'a$ *nd-fun* (-° [*101*] *100*) **where**
[*upred-defs*]: *uexpr-nd-fun* e = *Abs-nd-fun* $[\![e]\!]_e$

lift-definition *nd-fun-uexpr* :: $'a$ *nd-fun* $\Rightarrow$ $('a\ set,\ 'a)$ *uexpr* ($\text{-}_\circ$ [*101*] *100*) **is**
Rep-nd-fun **.**

no-utp-lift *nd-fun-uexpr*

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

update-uexpr-rep-eq-thms

lemma *uexpr-nd-fun-inverse* [*simp*]: $(P^\circ)_\circ = P$
  by (*pred-auto*)

lemma *nd-fun-ext*: $(\bigwedge x.\ (f_\bullet)\ x = (g_\bullet)\ x) \Longrightarrow f = g$
  apply(*subgoal-tac Rep-nd-fun* f = *Rep-nd-fun* g)

using *Rep-nd-fun-inject*
apply *blast*
by *blast*

lemma *nd-fun-eq-iff*: $(f = g) = (\forall x. (f_\bullet)\ x = (g_\bullet)\ x)$
by (*auto simp*: *nd-fun-ext*)

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

definition $0 = \zeta^\bullet$

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

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

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

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

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

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

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

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

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

end

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

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

lemma *nd-fun-n-op-one*[*nd-fun-aka*]: $n\ (n\ (1{::}'a\ \mathit{nd\text{-}fun})) = 1$
and *nd-fun-n-op-mult*[*nd-fun-aka*]: $n\ (n\ (n\ x \cdot n\ y)) = n\ x \cdot n\ y$
and *nd-fun-n-op-mult-comp*[*nd-fun-aka*]: $n\ x \cdot n\ (n\ x) = 0$
and *nd-fun-n-op-de-morgan*[*nd-fun-aka*]: $n\ (n\ (n\ x) \cdot n\ (n\ y)) = n\ x + n\ y$ **for** $x{::}'a\ \mathit{nd\text{-}fun}$
unfolding *n-op-nd-fun-def one-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*

by (*auto simp*: *nd-fun-eq-iff kcomp-def*)

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

end

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

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

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

end

end

3 Verification and refinement of HS in the state transformer KAT

We use our state transformers model to obtain verification and refinement components for hybrid programs. We devise three methods for reasoning with evolution commands and their continuous dynamics : providing flows, solutions or invariants.

theory *KAT-rKAT-rVCs-ndfun*
imports
KAT-rKAT-Models
Hybrid-Systems-VCs.HS-ODEs

begin recall-syntax

3.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

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

— Canonical lifting from predicates to state transformers and its simplification rules

definition *p2ndf* :: $'a\ upred \Rightarrow 'a\ nd\text{-}fun$ (($1\lceil$-$\rceil$))
 where $\lceil Q \rceil \equiv (\lambda\ x{::}'a.\ \{s{::}'a.\ s = x \wedge [\![Q]\!]_e\ s\})^\bullet$

lemma *p2ndf-simps*[*simp*]:
 $\lceil P \rceil \leq \lceil Q \rceil = `P \Rightarrow Q`$
 $(\lceil P \rceil = \lceil Q \rceil) = `P \Leftrightarrow Q`$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil P \wedge Q \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil P \vee Q \rceil$
 $t\ \lceil P \rceil = \lceil P \rceil$
 $n\ \lceil P \rceil = \lceil \neg\ P \rceil$
 unfolding *p2ndf-def one-nd-fun-def less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def*

 by (*auto simp add*: *nd-fun-eq-iff kcomp-def le-fun-def n-op-nd-fun-def conj-upred-def*

  inf-uexpr.rep-eq disj-upred-def sup-uexpr.rep-eq not-upred-def uminus-uexpr-def

  impl.rep-eq uexpr-appl.rep-eq lit.rep-eq taut.rep-eq iff-upred.rep-eq)

— Meaning of the state-transformer Hoare triple

lemma *ndfun-kat-H*: $H\ \lceil P \rceil\ X\ \lceil Q \rceil \longleftrightarrow (\forall\, s\ s'.\ [\![P]\!]_e\ s \longrightarrow s' \in (X_\bullet)\ s \longrightarrow [\![Q]\!]_e\ s')$
 unfolding *Hoare-def p2ndf-def less-eq-nd-fun-def times-nd-fun-def kcomp-def*
 by (*auto simp add*: *le-fun-def n-op-nd-fun-def*)

abbreviation *HTriple* (**{**-**}** - **{**-**}**) **where** $\mathbf{\{}P\mathbf{\}}X\mathbf{\{}Q\mathbf{\}} \equiv H\ \lceil P \rceil\ X\ \lceil Q \rceil$

utp-lift-notation *HTriple* (*0 2*)

— Hoare triple for skip and a simp-rule

abbreviation *skip* $\equiv$ (*1*::′*a nd-fun*)

lemma *H-skip*: $\{P\}$*skip*$\{P\}$
  using *H-skip* **by** *blast*

lemma *sH-skip*[*simp*]: $\{P\}$*skip*$\{Q\}$ $\longleftrightarrow$ ‘$P \Rightarrow Q$‘
  unfolding *ndfun-kat-H* **by** (*simp add*: *one-nd-fun-def impl.rep-eq taut.rep-eq*)

— Hoare logic consequence rule

lemma *H-conseq*:
  assumes $\{p_2\}S\{q_2\}$ ‘$p_1 \Rightarrow p_2$‘ ‘$q_2 \Rightarrow q_1$‘
  shows $\{p_1\}S\{q_1\}$
  using *assms*
  unfolding *ndfun-kat-H* **by** (*rel-auto*)

— We introduce assignments and compute derive their rule of Hoare logic.

definition *assigns* :: ′*s usubst* $\Rightarrow$ ′*s nd-fun* ($\langle$-$\rangle$) **where**
[*upred-defs*]: *assigns* $\sigma = (\lambda\ s.\ \{[\![\sigma]\!]_e\ s\})^\bullet$

abbreviation *assign* ((*2*- ::= -) [*70*, *65*] *61*)
  where *assign x e* $\equiv$ *assigns* [$\&x \mapsto_s e$]

utp-lift-notation *assign* (*1*)

lemma *H-assigns*: $P = (\sigma \dagger Q) \Longrightarrow \{P\}\ \langle\sigma\rangle\ \{Q\}$
  unfolding *ndfun-kat-H* **by** (*simp add*: *assigns-def*, *pred-auto*)

lemma *H-assign*: $P = Q[\![e/\&x]\!] \Longrightarrow \{P\}\ x ::= e\ \{Q\}$
  unfolding *ndfun-kat-H* **by** (*simp add*: *assigns-def*, *pred-auto*)

lemma *sH-assign*[*simp*]: $\{P\}\ x ::= e\ \{Q\} = (\forall\, s.\ [\![P]\!]_e\ s \longrightarrow [\![Q[\![e/\&x]\!]]\!]_e\ s)$
  unfolding *ndfun-kat-H* **by** (*pred-auto*)

lemma *sH-assigns*[*simp*]: $\{P\}\ \langle\sigma\rangle\ \{Q\} = (\forall\, s.\ [\![P]\!]_e\ s \longrightarrow [\![\sigma \dagger Q]\!]_e\ s)$
  unfolding *ndfun-kat-H* **by** (*pred-auto*)

lemma *sH-assign-alt*: $\{P\}x ::= e\{Q\}$ $\longleftrightarrow$ ‘$P \Rightarrow Q[\![e/x]\!]$‘
  unfolding *ndfun-kat-H* **by** (*pred-auto*)

lemma *H-assign-floyd-hoare*:
  assumes *vwb-lens x*
  shows $\{p\}\ x ::= e\ \{\exists\ v\ .\ p[\![\ll v\gg/x]\!] \wedge \&x = e[\![\ll v\gg/x]\!]\}$
  using *assms* **by** (*simp*, *rel-auto*′, *metis vwb-lens-wb wb-lens.get-put*)

— Next, the Hoare rule of the composition

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

lemma *H-seq*: $\{P\}\ X\ \{R\} \Longrightarrow \{R\}\ Y\ \{Q\} \Longrightarrow \{P\}\ X\ ;\ Y\ \{Q\}$
by (*auto intro*: *H-seq*)

lemma *sH-seq*: $\{P\}\ X\ ;\ Y\ \{Q\} = \{P\}\ X\ \{\forall\, s'.\ s' \in Y_\circ \Rightarrow Q[\![s'/\&\mathbf{v}]\!]\}$
unfolding *ndfun-kat-H* **by** (*auto simp*: *times-nd-fun-def kcomp-def*, *pred-auto+*)

lemma *H-seq-inv-1*: $\{P\}\ X\ \{P\} \Longrightarrow \{P\}\ Y\ \{Q\} \Longrightarrow \{P\}\ X\ ;\ Y\ \{Q\}$
by (*simp add*: *H-seq*)

lemma *H-seq-inv-2*: $\{P\}\ X\ \{Q\} \Longrightarrow \{Q\}\ Y\ \{Q\} \Longrightarrow \{P\}\ X\ ;\ Y\ \{Q\}$
by (*simp add*: *H-seq*)

Assignment laws

— Assignment forward law

lemma *H-assign-init*:
assumes *vwb-lens* x $\bigwedge x_0.\ \{\&x = e[\![\ll x_0\gg/\&x]\!] \wedge p[\![\ll x_0\gg/\&x]\!]\}S\{q\}$
shows $\{p\}(x ::= e)\ ;\ S\{q\}$
proof –
from *assms*(*2*) **have** $\{\exists\ v.\ p[\![v/x]\!] \wedge \&x = e[\![v/x]\!]\}\ S\ \{q\}$
unfolding *ndfun-kat-H* **by** (*rel-auto'*)
thus *?thesis*
by (*rule-tac H-seq*, *rule-tac H-assign-floyd-hoare*, *simp-all add*: *assms*)
qed

lemma *assign-self*: *vwb-lens* $x \Longrightarrow (x ::= \&x) = skip$
by (*rel-simp' simp*: *one-nd-fun.abs-eq*)

lemma *assigns-comp*: $\langle\sigma\rangle\ ;\ \langle\varrho\rangle = \langle\varrho \circ_s \sigma\rangle$
by (*simp add*: *assigns-def nd-fun-eq-iff subst-comp.rep-eq*, *transfer*, *simp add*: *kcomp-def*)

lemma *assign-twice*: *vwb-lens* $x \Longrightarrow (x ::= e)\ ;\ (x ::= f) = x ::= f[\![e/\&x]\!]$
by (*simp add*: *assigns-comp usubst*)

lemma *assign-commute*: $[\![\ x \bowtie y;\ x \sharp f;\ y \sharp e\ ]\!] \Longrightarrow (x ::= e)\ ;\ (y ::= f) = (y ::= f)\ ;\ (x ::= e)$
by (*simp add*: *assigns-comp usubst usubst-upd-comm*)

— Rewriting the Hoare rule for the conditional statement

abbreviation *cond-sugar* :: $'a$ *upred* $\Rightarrow$ $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* (*IF - THEN - ELSE -* [*64*,*64*] *63*)
  where *IF B THEN X ELSE Y* $\equiv$ *ifthenelse* $\lceil B \rceil$ *X Y*

utp-lift-notation *cond-sugar* (*0*)

lemma *H-cond*: $\{P \wedge B\}$ *X* $\{Q\} \Longrightarrow \{P \wedge \neg B\}$ *Y* $\{Q\} \Longrightarrow \{P\}$ *IF B THEN X ELSE Y* $\{Q\}$
  by (*rule H-cond*, *simp-all*)

lemma *sH-cond*[*simp*]: $\{P\}$ *IF B THEN X ELSE Y* $\{Q\}$ = ($\{P \wedge B\}$ *X* $\{Q\} \wedge$ $\{P \wedge \neg B\}$ *Y* $\{Q\}$)
  by (*auto simp*: *H-cond-iff ndfun-kat-H*)

lemma *assigns-test*: $\langle\sigma\rangle$; $\lceil p \rceil$ = $\lceil \sigma \dagger p \rceil$; $\langle\sigma\rangle$
  apply (*simp add*: *assigns-def n-op-nd-fun-def nd-fun-eq-iff subst-comp.rep-eq p2ndf-def*)
  apply (*transfer*)
  apply (*auto simp add*:*kcomp-def*)
  done

lemma *assigns-cond*:
  $\langle\sigma\rangle$; (*IF B THEN P ELSE Q*) = *IF* $\sigma \dagger B$ *THEN* $\langle\sigma\rangle$; *P ELSE* $\langle\sigma\rangle$; *Q*
  by (*simp add*: *ifthenelse-def KAT-rKAT-Models.nd-fun-distl assigns-test Groups.mult-ac*[*THEN sym*] *usubst*)

lemma *cond-assigns*: (*IF B THEN* $\langle\sigma\rangle$ *ELSE* $\langle\varrho\rangle$) = $\langle\sigma \triangleleft B \triangleright \varrho\rangle$
  apply (*simp add*: *ifthenelse-def assigns-def p2ndf-def n-op-nd-fun-def plus-nd-fun-def Abs-nd-fun-inject*)
  apply (*transfer*)
  apply (*auto simp add*: *kcomp-def sup-fun-def comp-def fun-eq-iff uIf-def*)
  done

lemmas *assign-simps* = *assigns-cond assigns-test assigns-comp*

— Rewriting the Hoare rule for the while loop

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

utp-lift-notation *while-inv-sugar* (*0*)

lemma *sH-while-inv*: ‘$P \Rightarrow I$‘ $\Longrightarrow$ ‘$I \wedge \neg B \Rightarrow Q$‘ $\Longrightarrow$ $\{I \wedge B\}$ *X* $\{I\}$
  $\Longrightarrow$ $\{P\}$ *WHILE B INV I DO X* $\{Q\}$
  by (*rule H-while-inv*, *simp-all add*: *ndfun-kat-H impl.rep-eq taut.rep-eq*)

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

abbreviation *loopi-sugar* :: *'a nd-fun* ⇒ *'a upred* ⇒ *'a nd-fun* (*LOOP - INV -* [*64*,*64*] *63*)
  where *LOOP X INV I* ≡ *loopi X* ⌈*I*⌉

utp-lift-notation *loopi-sugar* (*1*)

lemma *H-loop*: **{**P**}** X **{**P**}** ⟹ **{**P**}** *LOOP X INV I* **{**P**}**
  by (*auto intro*: *H-loop*)

lemma *H-loopI*: **{**I**}** X **{**I**}** ⟹ ⌈P⌉ ≤ ⌈I⌉ ⟹ ⌈I⌉ ≤ ⌈Q⌉ ⟹ **{**P**}** *LOOP X INV I* **{**Q**}**
  using *H-loop-inv*[*of* ⌈P⌉ ⌈I⌉ X ⌈Q⌉] **by** *auto*

3.2 Verification of hybrid programs

— Verification by providing evolution

definition *g-evol* :: ((*'a*::*ord*) ⇒ *'b usubst*) ⇒ *'b upred* ⇒ *'a set* ⇒ *'b nd-fun* (*EVOL*)
  where *EVOL* φ G T = ($\lambda s.$ *g-orbit* ($\lambda t.$ $[\![\varphi\ t]\!]_e$ s) $[\![G]\!]_e$ T)$^\bullet$

utp-lift-notation *g-evol* (*1*)

lemma *H-g-evol*:
  fixes φ :: (*'a*::*preorder*) ⇒ *'b usubst*
  assumes P = (∀ t∈«T» · (∀ τ∈«*down* T t» · $G[\![\varphi\ \tau/\&\mathbf{v}]\!]$) ⇒ $Q[\![\varphi\ t/\&\mathbf{v}]\!]$)
  shows **{**P**}** *EVOL* φ G T **{**Q**}**
  unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** (*simp add*: *assms*, *pred-auto*)

lemma *H-g-evol-alt*:
  fixes φ :: (*'a*::*preorder*) ⇒ *'b usubst*
  assumes P = (∀ t∈«T» · (∀ τ∈«*down* T t» · φ τ † G) ⇒ $Q[\![\varphi\ t/\&\mathbf{v}]\!]$)
  shows **{**P**}** *EVOL* φ G T **{**Q**}**
  using *assms* **by** (*rule-tac H-g-evol*, *pred-auto*)

lemma *sH-g-evol*[*simp*]:
  fixes φ :: (*'a*::*preorder*) ⇒ *'b usubst*
  shows **{**P**}** *EVOL* φ G T **{**Q**}** = ‘P ⇒ (∀ t∈«T» · (∀ τ∈«*down* T t» · $G[\![\varphi\ \tau/\&\mathbf{v}]\!]$) ⇒ $Q[\![\varphi\ t/\&\mathbf{v}]\!]$)‘
  unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** (*pred-auto*)

lemma *sH-g-evol-alt*[*simp*]:
  fixes φ :: (*'a*::*preorder*) ⇒ *'b usubst*
  shows **{**P**}** *EVOL* φ G T **{**Q**}** = ‘P ⇒ (∀ t∈«T» · (∀ τ∈«*down* T t» · φ τ † G) ⇒ φ t † Q)‘
  unfolding *ndfun-kat-H g-evol-def g-orbit-eq* **by** (*pred-auto*)

— Verification by providing solutions

definition *ivp-sols'* :: $((\prime a::real\text{-}normed\text{-}vector) \Rightarrow \prime a) \Rightarrow real\ set \Rightarrow \prime a\ set \Rightarrow real \Rightarrow ((real \Rightarrow \prime a)\ set,\ \prime a)\ uexpr$ **where**
[*upred-defs*]: $ivp\text{-}sols'\ \sigma\ T\ S\ t_0 = mk_e\ (ivp\text{-}sols\ (\lambda t.\ \sigma)\ T\ S\ t_0)$

definition *g-ode* $::((\prime a::banach) \Rightarrow \prime a) \Rightarrow \prime a\ upred \Rightarrow real\ set \Rightarrow \prime a\ set \Rightarrow real \Rightarrow \prime a\ nd\text{-}fun\ ((1x\acute{} = \text{-}\ \&\ \text{-}\ on\ \text{-}\ \text{-}\ @\ \text{-}))$
where $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0) \equiv (\lambda\ s.\ g\text{-}orbital\ f\ [\![G]\!]_e\ T\ S\ t_0\ s)^\bullet$

utp-lift-notation *g-ode* (*1*)

lemma *H-g-orbital*:
$P = (\forall\ X{\in}(\ll ivp\text{-}sols\ (\lambda\ t.\ f)\ T\ S\ t_0\gg\ |{>}\ \&\mathbf{v}) \cdot (\forall\ t{\in}\ll T\gg \cdot (\forall\ \tau \in \ll down\ T\ t\gg \cdot G[\![\ll X\ \tau\gg/\&\mathbf{v}]\!]) \Rightarrow Q[\![\ll X\ t\gg/\&\mathbf{v}]\!])) \Longrightarrow$
$\{P\}\ x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ \{Q\}$
unfolding *ndfun-kat-H g-ode-def g-orbital-eq* **by** *pred-simp*

lemma *sH-g-orbital*: $\{P\}\ x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ \{Q\} =$
$`P \Rightarrow (\forall\ X{\in}ivp\text{-}sols'\ f\ T\ S\ t_0 \cdot (\forall\ t{\in}\ll T\gg \cdot (\forall\ \tau \in \ll down\ T\ t\gg \cdot G[\![\ll X\ \tau\gg/\&\mathbf{v}]\!]) \Rightarrow Q[\![\ll X\ t\gg/\&\mathbf{v}]\!]))`$
unfolding *g-orbital-eq g-ode-def ndfun-kat-H* **by** (*pred-auto*)

locale *ue-local-flow* = *local-flow* $[\![\sigma]\!]_e\ T\ S\ \lambda\ t.\ [\![\varrho\ t]\!]_e$ **for** $\sigma\ \varrho\ T\ S$

context *local-flow*
begin

lemma *sH-g-ode*: $Hoare\ \lceil P\rceil\ (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall\ s{\in}S.\ [\![P]\!]_e\ s \longrightarrow (\forall\ t{\in}T.\ (\forall\ \tau{\in}down\ T\ t.\ [\![G]\!]_e\ (\varphi\ \tau\ s)) \longrightarrow [\![Q]\!]_e\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *rel-simp*, *safe*)
fix *s t*
assume *hyps*: $s \in S\ [\![P]\!]_e\ s\ t{\in}T\ \forall\tau.\ \tau \in T \wedge \tau \le t \longrightarrow [\![G]\!]_e\ (\varphi\ \tau\ s)$
and *main*: $\forall\ s.\ [\![P]\!]_e\ s \longrightarrow (\forall\ X.\ X{\in}Sols\ (\lambda t.\ f)\ T\ S\ 0\ s \longrightarrow (\forall\ t.\ t{\in}T \longrightarrow (\forall\tau.\ \tau \in T \wedge \tau \le t \longrightarrow [\![G]\!]_e\ (X\ \tau)) \longrightarrow [\![Q]\!]_e\ (X\ t)))$
hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s$
using *in-ivp-sols* **by** *blast*
thus $[\![Q]\!]_e\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*
next
fix *s X t*
assume *hyps*: $[\![P]\!]_e\ s\ X \in Sols\ (\lambda t.\ f)\ T\ S\ 0\ s\ t \in T\ \ \forall\tau.\ \tau \in T \wedge \tau \le t \longrightarrow [\![G]\!]_e\ (X\ \tau)$
and *main*: $\forall\ s{\in}S.\ [\![P]\!]_e\ s \longrightarrow (\forall\ t{\in}T.\ (\forall\tau.\ \tau \in T \wedge \tau \le t \longrightarrow [\![G]\!]_e\ (\varphi\ \tau\ s)) \longrightarrow [\![Q]\!]_e\ (\varphi\ t\ s))$
hence *obs*: $s \in S$

using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
hence $\forall\tau\in down\ T\ t.\ X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps* **by** *blast*
thus $[\![Q]\!]_e\ (X\ t)$
using *hyps main obs* **by** *auto*
qed

lemma *H-g-ode*:
assumes $P = (U(\&\mathbf{v} \in \ll S\gg) \Rightarrow (\forall\ t\in\ll T\gg \cdot (\forall\ \tau \in \ll down\ T\ t\gg \cdot G[\![\ll\varphi\ \tau\gg\ |\!>\ \&\mathbf{v}/\&\mathbf{v}]\!]) \Rightarrow Q[\![\ll\varphi\ t\gg\ |\!>\ \&\mathbf{v}/\&\mathbf{v}]\!]))$
shows *Hoare* $\lceil P\rceil\ (x' = f\ \&\ G\ on\ T\ S\ @\ 0)\ \lceil Q\rceil$
using *assms* **unfolding** *sH-g-ode* **by** *pred-simp*

lemma *sH-g-ode-ivl*: $\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow Hoare\ \lceil P\rceil\ (x' = f\ \&\ G\ on\ \{0..\tau\}\ S\ @\ 0)\ \lceil Q\rceil =$
$(\forall s\in S.\ [\![P]\!]_e\ s \longrightarrow (\forall t\in\{0..\tau\}.\ (\forall\tau\in\{0..t\}.\ [\![G]\!]_e\ (\varphi\ \tau\ s)) \longrightarrow [\![Q]\!]_e\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital*, *rel-simp*, *safe*)
fix *s t*
assume *hyps*: $0 \leq \tau\ \tau \in T\ s \in S\ [\![P]\!]_e\ s\ t \in \{0..\tau\}\ \forall\tau\in\{0..t\}.\ [\![G]\!]_e\ (\varphi\ \tau\ s)$
and *main*: $\forall s.\ [\![P]\!]_e\ s \longrightarrow (\forall X.\ X\in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s \longrightarrow (\forall t.\ 0 \leq t \wedge t \leq \tau \longrightarrow$
$(\forall\tau'.\ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow [\![G]\!]_e\ (X\ \tau')) \longrightarrow [\![Q]\!]_e\ (X\ t)))$
hence $(\lambda t.\ \varphi\ t\ s) \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s$
using *in-ivp-sols-ivl closed-segment-eq-real-ivl*[*of* $0\ \tau$] **by** *force*
thus $[\![Q]\!]_e\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*
next
fix *s X t*
assume *hyps*: $0 \leq \tau\ \tau \in T\ [\![P]\!]_e\ s\ X \in Sols\ (\lambda t.\ f)\ \{0..\tau\}\ S\ 0\ s\ 0 \leq t\ t \leq \tau$
$\forall\tau'.\ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow [\![G]\!]_e\ (X\ \tau')$
and *main*: $\forall s\in S.\ [\![P]\!]_e\ s \longrightarrow (\forall t\in\{0..\tau\}.\ (\forall\tau\in\{0..t\}.\ [\![G]\!]_e\ (\varphi\ \tau\ s)) \longrightarrow [\![Q]\!]_e\ (\varphi\ t\ s))$
hence $s \in S$
using *ivp-sols-def*[*of* $\lambda t.\ f$] *init-time* **by** *auto*
have *obs1*: $\forall\tau\in down\ \{0..\tau\}\ t.\ D\ X = (\lambda t.\ f\ (X\ t))\ on\ \{0--\tau\}$
apply(*clarsimp*, *rule has-vderiv-on-subset*)
using *ivp-solsD*(*1*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
have *obs2*: $X\ 0 = s\ \forall\tau\in down\ \{0..\tau\}\ t.\ X \in \{0--\tau\} \to S$
using *ivp-solsD*(*2*,*3*)[*OF hyps*(*4*)] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
have $\forall\tau\in down\ \{0..\tau\}\ t.\ \tau \in T$
using *subintervalI*[*OF init-time* ‹$\tau \in T$›] **by** (*auto simp*: *closed-segment-eq-real-ivl*)
hence $\forall\tau\in down\ \{0..\tau\}\ t.\ X\ \tau = \varphi\ \tau\ s$
using *obs1 obs2* **apply**(*clarsimp*)
by (*rule eq-solution-ivl*) (*auto simp*: *closed-segment-eq-real-ivl*)
thus $[\![Q]\!]_e\ (X\ t)$
using *hyps main* ‹$s \in S$› **by** *auto*
qed

lemma *H-g-ode-ivl*: $\tau \geq 0 \Longrightarrow \tau \in T \Longrightarrow$

$(\forall\, s{\in}S.\ \llbracket P\rrbracket_e\ s \longrightarrow (\forall\, t{\in}\{0..\tau\}.\ (\forall\, \tau{\in}\{0..t\}.\ \llbracket G\rrbracket_e\ (\varphi\ \tau\ s)) \longrightarrow \llbracket Q\rrbracket_e\ (\varphi\ t\ s)))$
$\Longrightarrow$
Hoare $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ \{0..\tau\}\ S\ @\ 0)$ $\lceil Q\rceil$
unfolding *sH-g-ode-ivl* **by** *simp*

lemma *H-g-ode-ivl2*:
assumes $P = (U(\&\mathbf{v} \in \ll S\gg) \Rightarrow (\forall\, t{\in}\ll\{0..\tau\}\gg \cdot (\forall\ \tau \in \ll\{0..t\}\gg \cdot G\llbracket\ll\varphi\ \tau\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}\rrbracket) \Rightarrow Q\llbracket\ll\varphi\ t\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}\rrbracket))$
and $\tau \geq 0$ **and** $\tau \in T$
shows *Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ \{0..\tau\}\ S\ @\ 0)$ $\lceil Q\rceil$
unfolding *sH-g-ode-ivl*[*OF assms*(*2*,*3*)] **using** *assms* **by** *pred-simp*

lemma *sH-orbit*: *Hoare* $\lceil P\rceil$ $(\gamma^{\varphi\bullet})$ $\lceil Q\rceil = (\forall\, s \in S.\ \llbracket P\rrbracket_e\ s \longrightarrow (\forall\ t \in T.\ \llbracket Q\rrbracket_e\ (\varphi\ t\ s)))$
using *sH-g-ode*[*of P true-upred Q*] **unfolding** *orbit-def g-ode-def* **by** *pred-simp*

end

— Verification with differential invariants

definition *g-ode-inv* :: $(('a{::}banach){\Rightarrow}'a) \Rightarrow 'a\ upred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ upred \Rightarrow 'a\ nd\text{-}fun$ ((*1x´=- & - on - - @ - DINV -*))
where $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) = (x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$

utp-lift-notation *g-ode-inv* (*1 5*)

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

lemma *sH-g-orbital-inv*:
assumes $\lceil P\rceil \leq \lceil I\rceil$ **and** *Hoare* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil$ **and** $\lceil I\rceil \leq \lceil Q\rceil$
shows *Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil Q\rceil$
using *assms*(*1*) **apply**(*rule-tac* $p'{=}\lceil I\rceil$ **in** *H-consl*, *simp*)
using *assms*(*3*) **apply**(*rule-tac* $q'{=}\lceil I\rceil$ **in** *H-consr*, *simp*)
using *assms*(*2*) **by** *simp*

lemma *sH-diff-inv*[*simp*]: *Hoare* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil$ = *diff-invariant* $\llbracket I\rrbracket_e\ f\ T\ S\ t_0\ \llbracket G\rrbracket_e$
unfolding *diff-invariant-eq ndfun-kat-H g-orbital-eq g-ode-def* **by** *auto*

lemma *H-g-ode-inv*: *Hoare* $\lceil I\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0)$ $\lceil I\rceil \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow$
$\lceil I \wedge G\rceil \leq \lceil Q\rceil \Longrightarrow$ *Hoare* $\lceil P\rceil$ $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I)$ $\lceil Q\rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $q'{=}\lceil I \wedge G\rceil$ **in** *H-consr*, *simp*)
apply(*subst sH-g-orbital-guard*[*of - G I*, *symmetric*], *pred-auto*)

by (*rule-tac I=I* **in** *sH-g-orbital-inv*, *simp-all*)

3.3 Refinement Components

abbreviation *RProgr* ([-,-]) **where** $[P,Q] \equiv Ref \lceil P\rceil \lceil Q\rceil$

utp-lift-notation *RProgr* (*0 1*)

— Skip

lemma *R-skip*: ‘$P \Rightarrow Q$‘ $\Longrightarrow$ *1* $\leq [P,Q]$
by (*auto simp*: *spec-def ndfun-kat-H one-nd-fun-def*, *pred-auto*)

— Composition

lemma *R-seq*: $[P,R]$; $[R,Q] \leq [P,Q]$
using *R-seq* **by** *blast*

lemma *R-seq-law*: $X \leq [P,R] \Longrightarrow Y \leq [R,Q] \Longrightarrow X;\ Y \leq [P,Q]$
unfolding *spec-def* **by** (*rule H-seq*)

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

— Assignment

lemma *R-assign*: $(x ::= e) \leq [P[\![e/\&x]\!],P]$
unfolding *spec-def* **by** (*rule H-assign*, *clarsimp simp*: *fun-eq-iff fun-upd-def*)

lemma *R-assign-law*: ‘$P \Rightarrow Q[\![e/\&x]\!]$‘ $\Longrightarrow (x ::= e) \leq [P,Q]$
unfolding *sH-assign*[*symmetric*] *spec-def* **by** (*metis pr-var-def sH-assign-alt*)

lemma *R-assignl*: $P = R[\![e/\&x]\!] \Longrightarrow (x ::= e)$; $[R,Q] \leq [P,Q]$
apply(*rule-tac R=R* **in** *R-seq-law*)
by (*rule-tac R-assign-law*, *simp-all*)

lemma *R-assignr*: $R = Q[\![e/\&x]\!] \Longrightarrow [P,R];\ (x ::= e) \leq [P,Q]$
apply(*rule-tac R=R* **in** *R-seq-law*, *simp*)
by (*rule-tac R-assign-law*, *simp*)

lemma $(x ::= e)$; $[Q,Q] \leq [Q[\![e/\&x]\!],Q]$
by (*rule R-assignl*) *simp*

lemma $[Q,Q[\![e/\&x]\!]]$; $(x ::= e) \leq [Q,Q]$
by (*rule R-assignr*) *simp*

— Conditional

lemma *R-cond*: $K1 = U(B \wedge P) \Longrightarrow K2 = U(\neg\ B \wedge P) \Longrightarrow$ (*IF B THEN* $[K1,Q]$ *ELSE* $[K2,Q]) \leq [P,Q]$

using *R-cond*[*of* $\lceil B \rceil$ $\lceil P \rceil$ $\lceil Q \rceil$] **by** *simp*

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

lemma *R-cond-law*: $X \leq [B \wedge P,Q] \Longrightarrow Y \leq [\neg B \wedge P,Q] \Longrightarrow$ (*IF B THEN X ELSE Y*) $\leq [P,Q]$
by (*rule order-trans*; (*rule R-cond-mono*)?, (*rule R-cond*)?) *auto*

— While loop

lemma *R-while*: $K=U(P \wedge \neg Q) \Longrightarrow$ *WHILE Q INV I DO* $[P \wedge Q,P] \leq [P,K]$
unfolding *while-inv-def* **using** *R-while*[*of* $\lceil Q \rceil$ $\lceil P \rceil$] **by** *simp*

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

lemma *R-while-law*: $X \leq [P \wedge B,P] \Longrightarrow Q = U(P \wedge \neg B) \Longrightarrow$ (*WHILE B INV I DO X*) $\leq [P,\ Q]$
by (*rule order-trans*; (*rule R-while-mono*)?, (*rule R-while*)?)

— Finite loop

lemma *R-loop*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow$ *LOOP* $[I,I]$ *INV I* $\leq$ $[P,Q]$
unfolding *spec-def* **by** (*rule H-loopI*, *rule R1*, *simp-all*)

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

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

— Evolution command (flow)

lemma *R-g-evol*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b usubst*
shows (*EVOL* φ *G T*) $\leq$ *Ref* $\lceil \forall\, t \in \ll T \gg \cdot (\forall\, \tau \in \ll down\ T\ t \gg \cdot \varphi\ \tau \dagger G) \Rightarrow \varphi\ t \dagger P \rceil$ $\lceil P \rceil$
unfolding *spec-def* **by** (*rule H-g-evol*, *rel-simp*)

lemma *R-g-evol-law*:
fixes φ :: ('*a*::*preorder*) $\Rightarrow$ '*b usubst*
shows '$P \Rightarrow (\forall\, t \in \ll T \gg \cdot (\forall\, \tau \in \ll down\ T\ t \gg \cdot \varphi\ \tau \dagger G) \Rightarrow \varphi\ t \dagger Q)$' $\Longrightarrow$ (*EVOL* φ *G T*) $\leq [P,Q]$
unfolding *sH-g-evol-alt*[*symmetric*] *spec-def* **by** (*auto*)

lemma *R-g-evoll*:
fixes φ :: $('a::preorder) \Rightarrow 'b$ *usubst*
shows $P = (\forall\, t{\in}\ll T\gg \cdot (\forall\, \tau{\in}\ll down\ T\ t\gg \cdot \varphi\ \tau \dagger G) \Rightarrow \varphi\ t \dagger R) \Longrightarrow$
$(EVOL\ \varphi\ G\ T)\ ;\ [R,Q] \leq [P,Q]$
apply(*rule-tac R=R* **in** *R-seq-law*)
by (*rule-tac R-g-evol-law*, *simp-all*)

lemma *R-g-evolr*:
fixes φ :: $('a::preorder) \Rightarrow 'b$ *usubst*
shows $R = (\forall\, t{\in}\ll T\gg \cdot (\forall\, \tau{\in}\ll down\ T\ t\gg \cdot \varphi\ \tau \dagger G) \Rightarrow \varphi\ t \dagger Q) \Longrightarrow$
$[P,R];\ (EVOL\ \varphi\ G\ T) \leq [P,Q]$
apply(*rule-tac R=R* **in** *R-seq-law*, *simp*)
by (*rule-tac R-g-evol-law*, *simp*)

lemma
fixes φ :: $('a::preorder) \Rightarrow 'b$ *usubst*
shows $EVOL\ \varphi\ G\ T\ ;\ [Q,Q] \leq Ref\ \lceil\forall\, t{\in}\ll T\gg \cdot (\forall\, \tau{\in}\ll down\ T\ t\gg \cdot \varphi\ \tau \dagger G)$
$\Rightarrow \varphi\ t \dagger Q\rceil\ \lceil Q\rceil$
by (*rule R-g-evoll*) *simp*

lemma
fixes φ :: $('a::preorder) \Rightarrow 'b$ *usubst*
shows $Ref\ \lceil Q\rceil\ \lceil\forall\, t{\in}\ll T\gg \cdot (\forall\, \tau{\in}\ll down\ T\ t\gg \cdot \varphi\ \tau \dagger G) \Rightarrow \varphi\ t \dagger Q\rceil\ ;\ EVOL$
$\varphi\ G\ T \leq [Q,Q]$
by (*rule R-g-evolr*) *simp*

— Evolution command (ode)

context *local-flow*
begin

lemma *R-g-ode*: $(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ 0) \leq Ref\ \lceil U(\&\mathbf{v}{\in}\ll S\gg \Rightarrow (\forall\, t{\in}\ll T\gg.$
$(\forall\, \tau{\in}\ll down\ T\ t\gg.\ G[\![\ll\varphi\ \tau\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}]\!]) \Rightarrow P[\![\ll\varphi\ t\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}]\!]))\rceil\ \lceil P\rceil$
unfolding *spec-def* **by** (*rule H-g-ode*, *rel-auto*)

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

lemma *R-g-odel*: $P = U(\forall\, t{\in}\ll T\gg.\ (\forall\, \tau{\in}\ll down\ T\ t\gg.\ G[\![\ll\varphi\ \tau\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}]\!]) \longrightarrow$
$R[\![\ll\varphi\ t\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}]\!]) \Longrightarrow$
$(x\acute{}= f\ \&\ G\ on\ T\ S\ @\ 0)\ ;\ Ref\ \lceil R\rceil\ \lceil Q\rceil \leq [P,Q]$
apply(*rule-tac R=R* **in** *R-seq-law*)
apply (*rule-tac R-g-ode-law*, *simp-all*, *rel-auto*)
done

lemma *R-g-oder*: $R = U(\forall\, t{\in}\ll T\gg.\ (\forall\, \tau{\in}\ll down\ T\ t\gg.\ G[\![\ll\varphi\ \tau\gg\ |{>}\ \&\mathbf{v}/\&\mathbf{v}]\!]) \longrightarrow$

$Q[\![\ll\varphi\ t\gg\ |>\ \&\mathbf{v}/\&\mathbf{v}]\!]) \Longrightarrow$
  $[P,R]$; $(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq [P,Q]$
  apply(*rule-tac R=R* **in** *R-seq-law*, *simp*)
  by (*rule-tac R-g-ode-law*, *rel-simp*)

lemma $(x' = f\ \&\ G\ on\ T\ S\ @\ 0)$; $[Q,Q] \leq Ref\ \lceil U(\forall t{\in}{\ll}T{\gg}.\ (\forall \tau{\in}{\ll}down\ T\ t{\gg}.\ G[\![\ll\varphi\ \tau\gg\ |>\ \&\mathbf{v}/\&\mathbf{v}]\!]) \longrightarrow Q[\![\ll\varphi\ t\gg\ |>\ \&\mathbf{v}/\&\mathbf{v}]\!])\rceil\ \lceil Q\rceil$
  by (*rule R-g-odel*) *simp*

lemma $Ref\ \lceil Q\rceil\ \lceil U(\forall t{\in}{\ll}T{\gg}.\ (\forall \tau{\in}{\ll}down\ T\ t{\gg}.\ G[\![\ll\varphi\ \tau\gg\ |>\ \&\mathbf{v}/\&\mathbf{v}]\!]) \Rightarrow Q[\![\ll\varphi\ t\gg\ |>\ \&\mathbf{v}/\&\mathbf{v}]\!])\rceil$; $(x' = f\ \&\ G\ on\ T\ S\ @\ 0) \leq [Q,Q]$
  by (*rule R-g-oder*) *rel-simp*

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

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $[\![I]\!]_e\ f\ T\ S\ t_0\ [\![G]\!]_e \Longrightarrow \lceil P\rceil \leq \lceil I\rceil \Longrightarrow \lceil I \wedge G\rceil \leq \lceil Q\rceil \Longrightarrow$
  $(x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0\ DINV\ I) \leq [P,Q]$
  unfolding *spec-def* **by** (*auto simp*: *H-g-ode-inv*)

3.4 Derivation of the rules of dL

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

lemma *diff-solve-axiom*:
  fixes $c{::}'a{::}\{heine\text{-}borel,\ banach\}$
  assumes $0 \in T$ **and** *is-interval T open T*
    and $\forall s.\ [\![P]\!]_e\ s \longrightarrow (\forall t{\in}T.\ (\mathcal{P}\ (\lambda\ t.\ s + t *_R c)\ (down\ T\ t) \subseteq \{s.\ [\![G]\!]_e\ s\}) \longrightarrow [\![Q]\!]_e\ (s + t *_R c))$
  shows *Hoare* $\lceil P\rceil$ $(x'{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)$ $\lceil Q\rceil$
  apply(*subst local-flow.sH-g-ode*[**where** $f{=}\lambda s.\ c$ **and** $\varphi{=}(\lambda\ t\ x.\ x + t *_R c)$])
  using *line-is-local-flow assms* **by** *auto*

lemma *diff-solve-rule*:
  assumes *local-flow f T UNIV* φ
    and $\forall s.\ [\![P]\!]_e\ s \longrightarrow (\forall\ t{\in}T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ [\![G]\!]_e\ s\}) \longrightarrow [\![Q]\!]_e\ (\varphi\ t\ s))$
  shows *Hoare* $\lceil P\rceil$ $(x' = f\ \&\ G\ on\ T\ UNIV\ @\ 0)$ $\lceil Q\rceil$
  using *assms* **by**(*subst local-flow.sH-g-ode*, *auto*)

lemma *diff-weak-rule*:

assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows *Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ t_0)$ $\lceil Q \rceil$
using *assms* **unfolding** *ndfun-kat-H g-ode-def g-orbital-eq ivp-sols-def* **by** (*simp*, *rel-auto*)

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and *wp-C*:*Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ t_0)$ $\lceil C \rceil$
and *wp-Q*:*Hoare* $\lceil P \rceil$ $(x' = f \ \& \ (G \wedge C) \ on \ T \ S \ @ \ t_0)$ $\lceil Q \rceil$
shows *Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ on \ T \ S \ @ \ t_0)$ $\lceil Q \rceil$
proof(*subst ndfun-kat-H*, *simp add*: *g-orbital-eq p2ndf-def g-ode-def*, *clarsimp*)
fix t::*real* **and** X::*real* $\Rightarrow$ *'a* **and** s **assume** $[\![P]\!]_e$ s **and** $t \in T$
and *x-ivp*:$X \in$ *ivp-sols* $(\lambda t.\ f)$ T S t_0 s
and *guard-x*:$\forall x.\ x \in T \wedge x \leq t \longrightarrow [\![G]\!]_e$ $(X\ x)$
have $\forall t \in (down\ T\ t).\ X\ t \in$ *g-orbital* f $[\![G]\!]_e$ T S t_0 s
using *g-orbitalI*[*OF x-ivp*] *guard-x* **by** *auto*
hence $\forall t \in (down\ T\ t).\ [\![C]\!]_e$ $(X\ t)$
using *wp-C* ‹$[\![P]\!]_e$ s› **by** (*subst* (*asm*) *ndfun-kat-H*, *auto simp*: *g-ode-def*)
hence $X\ t \in$ *g-orbital* f $[\![G \wedge C]\!]_e$ T S t_0 s
using *guard-x* ‹$t \in T$› **by** (*auto intro*!: *g-orbitalI x-ivp*, *rel-simp*)
thus $[\![Q]\!]_e$ $(X\ t)$
using ‹$[\![P]\!]_e$ s› *wp-Q* **by** (*subst* (*asm*) *ndfun-kat-H*) (*auto simp*: *g-ode-def*)
qed

abbreviation *g-global-ode* ::((*'a*::*banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a upred* $\Rightarrow$ *'a nd-fun* ((*1x'*=- & -))
where $(x' = f \ \& \ G) \equiv (x' = f \ \& \ G \ on \ UNIV \ UNIV \ @ \ 0)$

utp-lift-notation *g-global-ode* (*1*)

abbreviation *g-global-ode-inv* :: ((*'a*::*banach*)$\Rightarrow$*'a*) $\Rightarrow$ *'a upred* $\Rightarrow$ *'a upred* $\Rightarrow$ *'a nd-fun*
((*1x'*=- & - *DINV* -)) **where** $(x' = f \ \& \ G \ DINV \ I) \equiv (x' = f \ \& \ G \ on \ UNIV \ UNIV \ @ \ 0 \ DINV \ I)$

utp-lift-notation *g-global-ode-inv* (*1 2*)

end

3.5 Examples

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

theory *KAT-rKAT-Examples-ndfun*
imports *KAT-rKAT-rVCs-ndfun*

begin

declare [[*coercion Rep-uexpr*]]

— Lens definition for examples

utp-lit-vars

definition *vec-lens* :: $'i \Rightarrow ('a \Longrightarrow 'a\hat{}'i)$ **where**
[*lens-defs*]: *vec-lens* $k = (\!|$ *lens-get* $= (\lambda\ s.\ vec\text{-}nth\ s\ k)$
$\quad ,\ lens\text{-}put = (\lambda\ s\ v.\ (\chi\ j.\ ((((\$)\ s)(k := v))\ j))\ |\!)$

lemma *vec-vwb-lens* [*simp*]: *vwb-lens* (*vec-lens* *k*)
  apply (*unfold-locales*)
  apply (*simp-all add*: *vec-lens-def fun-eq-iff*)
  using *vec-lambda-unique* **apply** *fastforce*
  done

lemma *vec-lens-indep* [*simp*]: $(i \neq j) \Longrightarrow (vec\text{-}lens\ i \bowtie vec\text{-}lens\ j)$
  by (*simp add*: *lens-indep-vwb-iff*, *auto simp add*: *lens-defs*)

— A tactic for verification of hybrid programs

named-theorems *hoare-intros*

declare *H-assign-init* [*hoare-intros*]
  and *H-cond* [*hoare-intros*]
  and *local-flow.H-g-ode-ivl* [*hoare-intros*]
  and *H-g-ode-inv* [*hoare-intros*]

method *body-hoare*
  = (*rule hoare-intros*,(*simp*)?; *body-hoare?*)

method *hyb-hoare* **for** P::$'a$ *upred*
  = (*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-loop-law* [*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*)?)

declare *forall-2* [*simp*]
    and *forall-3* [*simp*]
    and *forall-4* [*simp*]

3.5.1 Pendulum

abbreviation x :: *real* $\Longrightarrow$ *real*ˆ*2* **where** $x \equiv$ *vec-lens 1*
abbreviation y :: *real* $\Longrightarrow$ *real*ˆ*2* **where** $y \equiv$ *vec-lens 2*

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 prove that this motion remains circular.

abbreviation *fpend* :: (*real*ˆ*2*) *usubst* (f)
  where *fpend* $\equiv [x \mapsto_s y,\ y \mapsto_s -x]$

abbreviation *pend-flow* :: *real* $\Rightarrow$ (*real*ˆ*2*) *usubst* (φ)
  where *pend-flow* $\tau \equiv [x \mapsto_s x \cdot cos\ \tau + y \cdot sin\ \tau,\ y \mapsto_s -\ x \cdot sin\ \tau + y \cdot cos\ \tau]$

— Verified with annotated dynamics

lemma *pendulum-dyn*: $\{r^2 = x^2 + y^2\}(EVOL\ \varphi\ G\ T)\{r^2 = x^2 + y^2\}$
  by (*simp*, *rel-auto*)

— Verified with invariants

lemma *pendulum-inv*: $\{r^2 = x^2 + y^2\}$ ($x\acute{}= f$ & G) $\{r^2 = x^2 + y^2\}$
  by (*simp*, *pred-simp*, *auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified by providing solutions

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

lemma *pendulum-flow*: $\{r^2 = x^2 + y^2\}$ ($x\acute{}= f$ & G) $\{r^2 = x^2 + y^2\}$
  by (*simp only*: *local-flow.sH-g-ode*[*OF local-flow-pend*], *pred-simp*)

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

3.5.2 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, thus it is a completely elastic collision with the ground. We prove that the ball remains above ground and below its initial resting position.

abbreviation v :: *real* $\Longrightarrow$ *real*ˆ*2*
  where $v \equiv$ *vec-lens 2*

abbreviation *fball* :: *real* $\Rightarrow$ (*real*, *2*) *vec* $\Rightarrow$ (*real*, *2*) *vec* (f)
  where $f\ g \equiv [x \mapsto_s v,\ v \mapsto_s g]$

abbreviation *ball-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*ˆ*2*) *usubst* (φ)
  where $\varphi\ g\ \tau \equiv [x \mapsto_s g \cdot \tau\ \hat{}\ 2/2 + v \cdot \tau + x,\ \ v \mapsto_s g \cdot \tau + v]$

— Verified with invariants

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

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

lemma *fball-invariant*:
  fixes $g\ h$ :: *real*
  defines *dinv*: $I \equiv \mathbf{U}(2 \cdot «g» \cdot x - 2 \cdot «g» \cdot «h» - (v \cdot v) = 0)$
  shows *diff-invariant I* ($f\ g$) *UNIV UNIV 0 G*
  unfolding *dinv* **apply**(*pred-simp*, *rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
  by(*auto intro*!: *poly-derivatives*)

abbreviation *bb-dinv* $g\ h \equiv$
  (*LOOP*

$((x' = f\ g\ \&\ (x \geq 0)\ DINV\ (2 \cdot g \cdot x - 2 \cdot g \cdot h - v \cdot v = 0));$
$(IF\ (v = 0)\ THEN\ (v ::= -v)\ ELSE\ skip))$
$INV\ (0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v))$

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow \{x = h \wedge v = 0\}\ bb\text{-}dinv\ g\ h\ \{0 \leq x \wedge x \leq h\}$
apply(*hyb-hoare* $\mathbf{U}(0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v)$)
using *fball-invariant* **by** (*simp-all*, *rel-auto' simp*: *bb-real-arith*)

— Verified with annotated dynamics

lemma [*bb-real-arith*]:
fixes *x v* :: *real*
assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$
shows $2 \cdot g \cdot h + (-(g \cdot \tau) - v) \cdot (-(g \cdot \tau) - v) = 0$
and $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
from *pos* **have** $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
then have $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$
by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right*
monoid-mult-class.power2-eq-square semiring-class.distrib-left)
hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot h = 0$
using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

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

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

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

abbreviation *bb-evol g h T* ≡
LOOP
EVOL (φ *g*) ($x \geq 0$) *T*;
(*IF* ($v = 0$) *THEN* ($v ::= -v$) *ELSE skip*)
INV ($0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$)

lemma *bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows **{**$x = h \wedge v = 0$**}** *bb-evol g h T* **{**$0 \leq x \wedge x \leq h$**}**
apply(*hyb-hoare* **U**($0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$))
using *assms* **by** (*rel-auto′ simp*: *bb-real-arith*)

— Verified by providing solutions

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

abbreviation *bb-sol g h* ≡
(*LOOP* (
($x´= f\ g$ & ($x \geq 0$));
(*IF* ($v = 0$) *THEN* ($v ::= -v$) *ELSE skip*))
INV ($0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$))

lemma *bouncing-ball-flow*:
assumes $g < 0$ **and** $h \geq 0$
shows **{**$x = h \wedge v = 0$**}** *bb-sol g h* **{**$0 \leq x \wedge x \leq h$**}**
apply(*hyb-hoare* **U**($0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$))
apply(*subst local-flow.sH-g-ode*[*OF local-flow-ball*])
using *assms* **by** (*rel-auto′ simp*: *bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g <$ (*0*::*real*) $\Longrightarrow 0 \leq h \Longrightarrow$
[$v = 0 \wedge 0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$, $0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$] $\geq$ ($v ::= -\ v$)
by (*rule R-assign-law*, *pred-simp*)

lemma *R-bouncing-ball-dyn*:
  assumes $g < 0$ **and** $h \geq 0$
  shows $[x = h \wedge v = 0,\ 0 \leq x \wedge x \leq h] \geq$ *bb-evol g h T*
  apply(*refinement*; (*rule R-bb-assign*[*OF assms*])?)
  using *assms* **by** (*rel-auto′ simp*: *bb-real-arith*)

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

3.5.3 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 $T' = -\ a * (T - c)$ where $c = L \geq 0$ when the heater is on, and $c = 0$ when it is off. We prove that the thermostat keeps the room's temperature between T_l and T_h.

hide-const t

abbreviation T :: *real* $\Longrightarrow$ *real^4* **where** $T \equiv$ *vec-lens 1*
abbreviation t :: *real* $\Longrightarrow$ *real^4* **where** $t \equiv$ *vec-lens 2*
abbreviation T_0 :: *real* $\Longrightarrow$ *real^4* **where** $T_0 \equiv$ *vec-lens 3*
abbreviation ϑ :: *real* $\Longrightarrow$ *real^4* **where** $\vartheta \equiv$ *vec-lens 4*

abbreviation *ftherm* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*, *4*) *vec* $\Rightarrow$ (*real*, *4*) *vec* (f)
  where $f\ a\ c \equiv [T \mapsto_s -\ (a * (T - c)),\ T_0 \mapsto_s 0,\ \vartheta \mapsto_s 0,\ t \mapsto_s 1]$

abbreviation *therm-guard* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *upred* (G)
  where $G\ T_l\ T_h\ a\ L \equiv \mathbf{U}(t \leq -\ (ln\ ((L-(\mathit{if}\ L{=}0\ \mathit{then}\ T_l\ \mathit{else}\ T_h))/(L-T_0)))/a)$

no-utp-lift *therm-guard* (*0 1 2 3*)

abbreviation *therm-loop-inv* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *upred* (I)
  where $I\ T_l\ T_h \equiv \mathbf{U}(T_l \leq T \wedge T \leq T_h \wedge (\vartheta = 0 \vee \vartheta = 1))$

no-utp-lift *therm-loop-inv* (*0 1*)

abbreviation *therm-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *usubst* (φ)
  where $\varphi\ a\ c\ \tau \equiv [T \mapsto_s -\ exp(-a * \tau) * (c - T) + c,\ t \mapsto_s \tau + t,\ T_0 \mapsto_s T_0,\ \vartheta \mapsto_s \vartheta]$

abbreviation *therm-ctrl* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *nd-fun* (*ctrl*)
  where *ctrl* $T_l\ T_h \equiv$
  $(t ::= 0)$; $(T_0 ::= T)$;
  (*IF* $(\vartheta = 0 \wedge T_0 \leq T_l + 1)$ *THEN* $(\vartheta ::= 1)$ *ELSE*
   IF $(\vartheta = 1 \wedge T_0 \geq T_h - 1)$ *THEN* $(\vartheta ::= 0)$ *ELSE skip*)

abbreviation *therm-dyn* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *nd-fun*

(*dyn*)
  where *dyn* T_l T_h *a* T_u $\tau \equiv$
  IF $(\vartheta = 0)$ *THEN* $x' = f\ a\ 0\ \&\ G\ T_l\ T_h\ a\ 0$ *on* $\{0..\tau\}$ *UNIV* @ *0*
   ELSE $x' = f\ a\ T_u\ \&\ G\ T_l\ T_h\ a\ T_u$ *on* $\{0..\tau\}$ *UNIV* @ *0*

abbreviation *therm* T_l T_h *a L* $\tau \equiv$ *LOOP* (*ctrl* T_l T_h ; *dyn* T_l T_h *a L* τ) *INV* (*I* T_l T_h)

— Verified by providing solutions

lemma *norm-diff-therm-dyn*: $0 < (a::real) \Longrightarrow (a \cdot (s_2\$1 - T_u) - a \cdot (s_1\$1 - T_u))^2$
  $\leq (a \cdot sqrt\ ((s_1\$1 - s_2\$1)^2 + ((s_1\$2 - s_2\$2)^2 + ((s_1\$3 - s_2\$3)^2 + (s_1\$4 - s_2\$4)^2))))^2$
proof(*simp add*: *field-simps*)
  assume *a1*: $0 < a$
  have $(a \cdot s_2\$1 - a \cdot s_1\$1)^2 = a^2 \cdot (s_2\$1 - s_1\$1)^2$
    by (*metis* (*mono-tags*, *hide-lams*) *Rings.ring-distribs*(*4*) *mult.left-commute semiring-normalization-rules*(*18*) *semiring-normalization-rules*(*29*))
  moreover have $(s_2\$1 - s_1\$1)^2 \leq (s_1\$1 - s_2\$1)^2 + ((s_1\$2 - s_2\$2)^2 + ((s_1\$3 - s_2\$3)^2 + (s_1\$4 - s_2\$4)^2))$
    using *zero-le-power2* **by** (*simp add*: *power2-commute*)
  thus $(a \cdot s_2\ \$\ 1 - a \cdot s_1\ \$\ 1)^2 \leq a^2 \cdot (s_1\ \$\ 1 - s_2\ \$\ 1)^2 +$
  $(a^2 \cdot (s_1\ \$\ 2 - s_2\ \$\ 2)^2 + (a^2 \cdot (s_1\ \$\ 3 - s_2\ \$\ 3)^2 + a^2 \cdot (s_1\ \$\ 4 - s_2\ \$\ 4)^2))$
    using *a1* **by** (*simp add*: *Groups.algebra-simps*(*18*)[*symmetric*] *calculation*)
qed

lemma *local-lipschitz-therm-dyn*:
  assumes $0 < (a::real)$
  shows *local-lipschitz UNIV UNIV* ($\lambda t::real.\ f\ a\ T_u$)
  apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
  apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*, *pred-simp*)
  using *assms* **apply**(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *pred-simp*)
  unfolding *real-sqrt-abs*[*symmetric*] **apply** (*rule real-le-lsqrt*)
  by (*simp-all add*: *norm-diff-therm-dyn*)

lemma *local-flow-therm*: $a > 0 \Longrightarrow$ *local-flow* ($f\ a\ T_u$) *UNIV UNIV* ($\varphi\ a\ T_u$)
  apply (*unfold-locales*, *simp-all*)
  using *local-lipschitz-therm-dyn* **apply**(*pred-simp*)
  by (*pred-simp*, *force intro*!: *poly-derivatives simp*: *vec-eq-iff*)+

lemma *therm-dyn-down*:
  fixes *T*::*real*
  assumes $a > 0$ **and** *Thyps*: $0 < T_l$ $T_l \leq T$ $T \leq T_h$
    and *thyps*: $0 \leq (\tau::real)$ $\forall\tau\in\{0..\tau\}.\ \tau \leq -(ln\ (T_l\ /\ T)\ /\ a)$
  shows $T_l \leq exp\ (-a * \tau) * T$ **and** $exp\ (-a * \tau) * T \leq T_h$
proof−
  have $0 \leq \tau \wedge \tau \leq -(ln\ (T_l\ /\ T)\ /\ a)$
    using *thyps* **by** *auto*

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

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

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

lemma *thermostat-flow*:
assumes *0* < *a* **and** *0* $\leq \tau$ **and** *0* < T_l **and** $T_h < T_u$
shows **{**$I\ T_l\ T_h$**}** *therm* T_l T_h *a* T_u τ **{**$I\ T_l\ T_h$**}**
apply(*hyb-hoare* **U**($I\ T_l\ T_h$ $\wedge$ *t*=*0* $\wedge$ T_0 = *T*))
prefer *4* **prefer** *8* **using** *local-flow-therm assms* **apply** *force*+

using *assms therm-dyn-up therm-dyn-down* **by** *rel-auto′*

— Refined by providing solutions

lemma *R-therm-down*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $\lceil \vartheta = 0 \wedge I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T,\ I\ T_l\ T_h \rceil \geq$
$(x' = f\ a\ 0\ \&\ G\ T_l\ T_h\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *therm-dyn-down*[*OF assms*(*1*,*3*), *of* - T_h] *assms* **by** *rel-auto′*

lemma *R-therm-up*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $\lceil \neg\ \vartheta = 0 \wedge I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T,\ I\ T_l\ T_h \rceil \geq$
$(x' = f\ a\ T_u\ \&\ G\ T_l\ T_h\ a\ T_u\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *therm-dyn-up*[*OF assms*(*1*) - - *assms*(*4*), *of* T_l] *assms* **by** *rel-auto′*

lemma *R-therm-time*: $\lceil I\ T_l\ T_h,\ I\ T_l\ T_h \wedge t = 0 \rceil \geq (t ::= 0)$
by (*rule R-assign-law*, *pred-simp*)

lemma *R-therm-temp*: $\lceil I\ T_l\ T_h \wedge t = 0,\ I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T \rceil \geq (T_0 ::= T)$
by (*rule R-assign-law*, *pred-simp*)

lemma *R-thermostat-flow*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $\lceil I\ T_l\ T_h,\ I\ T_l\ T_h \rceil \geq$ *therm* $T_l\ T_h\ a\ T_u\ \tau$
by (*refinement*; (*rule R-therm-time*)?, (*rule R-therm-temp*)?, (*rule R-assign-law*)?,

(*rule R-therm-up*[*OF assms*])?, (*rule R-therm-down*[*OF assms*])?) *rel-auto′*

no-notation *ftherm* (f)
and *therm-flow* (φ)
and *therm-guard* (G)
and *therm-loop-inv* (I)
and *therm-ctrl* (*ctrl*)
and *therm-dyn* (*dyn*)

3.5.4 Water tank

— Variation of Hespanha and [1]

abbreviation h :: *real* ⟹ *real*ˆ*4* **where** $h \equiv$ *vec-lens 1*
abbreviation h_0 :: *real* ⟹ *real*ˆ*4* **where** $h_0 \equiv$ *vec-lens 3*
abbreviation π :: *real* ⟹ *real*ˆ*4* **where** $\pi \equiv$ *vec-lens 4*

abbreviation *ftank* :: *real* ⇒ (*real*, *4*) *vec* ⇒ (*real*, *4*) *vec* (f)
where $f\ k \equiv [\pi \mapsto_s 0,\ h \mapsto_s k,\ h_0 \mapsto_s 0,\ t \mapsto_s 1]$

abbreviation *tank-flow* :: *real* ⇒ *real* ⇒ (*real^4*) *usubst* (φ)
  where $\varphi\ k\ \tau \equiv [h \mapsto_s k * \tau + h,\ t \mapsto_s \tau + t,\ h_0 \mapsto_s h_0,\ \pi \mapsto_s \pi]$

abbreviation *tank-guard* :: *real* ⇒ *real* ⇒ (*real^4*) *upred* (G)
  where $G\ h_x\ k \equiv \mathbf{U}(t \leq (h_x - h_0)/k)$

no-utp-lift *tank-guard* (*0 1*)

abbreviation *tank-loop-inv* :: *real* ⇒ *real* ⇒ (*real^4*) *upred* (I)
  where $I\ h_l\ h_h \equiv \mathbf{U}(h_l \leq h \wedge h \leq h_h \wedge (\pi = 0 \vee \pi = 1))$

no-utp-lift *tank-loop-inv* (*0 1*)

abbreviation *tank-diff-inv* :: *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *upred* (dI)
  where $dI\ h_l\ h_h\ k \equiv \mathbf{U}(h = k \cdot t + h_0 \wedge 0 \leq t \wedge h_l \leq h_0 \wedge h_0 \leq h_h \wedge (\pi = 0$
$\vee\ \pi = 1))$

no-utp-lift *tank-diff-inv* (*0 1 2*)

— Verified by providing solutions

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

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

abbreviation *tank-ctrl* :: *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (*ctrl*)
  where *ctrl* $h_l\ h_h \equiv (t ::= 0);(h_0 ::= h)$;
  (*IF* $(\pi = 0 \wedge h_0 \leq h_l + 1)$ *THEN* $(\pi ::= 1)$ *ELSE*
  (*IF* $(\pi = 1 \wedge h_0 \geq h_h - 1)$ *THEN* $(\pi ::= 0)$ *ELSE skip*))

abbreviation *tank-dyn-sol* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (*dyn*)
  where *dyn* $c_i\ c_o\ h_l\ h_h\ \tau \equiv$ (*IF* $(\pi = 0)$ *THEN*
    $(x' = f\ (c_i - c_o)\ \&\ G\ h_h\ (c_i - c_o)$ *on* $\{0..\tau\}$ *UNIV* @ $0)$

ELSE $(x' = f\ (-c_o)\ \&\ G\ h_l\ (-c_o)$ *on* $\{0..\tau\}$ *UNIV* @ *0*))

abbreviation *tank-sol* $c_i\ c_o\ h_l\ h_h\ \tau \equiv$ *LOOP* (*ctrl* $h_l\ h_h$; *dyn* $c_i\ c_o\ h_l\ h_h\ \tau$) *INV* ($I\ h_l\ h_h$)

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 $\{I\ h_l\ h_h\}$ *tank-sol* $c_i\ c_o\ h_l\ h_h\ \tau$ $\{I\ h_l\ h_h\}$
  apply(*hyb-hoare* $\mathbf{U}(I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h)$)
        prefer *4* **prefer** *8* **using** *assms local-flow-tank* **apply** *force+*
  using *assms tank-arith* **by** *rel-auto'*

no-notation *tank-dyn-sol* (*dyn*)

— Verified with invariants

lemma *tank-diff-inv*:
  $0 \leq \tau \Longrightarrow$ *diff-invariant* (*dI* $h_l\ h_h\ k$) (*f k*) $\{0..\tau\}$ *UNIV 0 Guard*
  apply(*pred-simp*, *intro diff-invariant-conj-rule*)
    apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
   apply(*rule-tac* $\nu' = \lambda t.\ 0$ **and** $\mu' = \lambda t.\ 1$ **in** *diff-invariant-leq-rule*, *simp-all*)
  apply(*rule-tac* $\nu' = \lambda t.\ 0$ **and** $\mu' = \lambda t.\ 0$ **in** *diff-invariant-leq-rule*, *simp-all*)
  by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

lemma *tank-inv-arith1*:
  assumes $0 \leq (\tau{::}real)$ **and** $c_o < c_i$ **and** b: $h_l \leq y_0$ **and** g: $\tau \leq (h_h - y_0) / (c_i - c_o)$
  shows $h_l \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq h_h$
proof−
  have $(c_i - c_o) \cdot \tau \leq (h_h - y_0)$
    using *g assms*(*2*,*3*) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
  thus $(c_i - c_o) \cdot \tau + y_0 \leq h_h$
    by *auto*
  show $h_l \leq (c_i - c_o) \cdot \tau + y_0$
    using *b assms*(*1*,*2*) **by** (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)
qed

lemma *tank-inv-arith2*:
  assumes $0 \leq (\tau{::}real)$ **and** $0 < c_o$ **and** b: $y_0 \leq h_h$ **and** g: $\tau \leq -((h_l - y_0) / c_o)$
  shows $h_l \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq h_h$
proof−
  have $\tau \cdot c_o \leq y_0 - h_l$
    using *g* ‹$0 < c_o$› *pos-le-minus-divide-eq* **by** *fastforce*
  thus $h_l \leq y_0 - c_o \cdot \tau$
    by (*auto simp*: *mult.commute*)

show $y_0 - c_o \cdot \tau \le h_h$
using *b assms*(*1*,*2*) **by** (*smt linordered-field-class.sign-simps*(*39*) *mult-less-cancel-right*)

qed

abbreviation *tank-dyn-dinv* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*ˆ*4*) *nd-fun* (*dyn*)
where $dyn\ c_i\ c_o\ h_l\ h_h\ \tau \equiv IF\ (\pi = 0)\ THEN$
$x' = f\ (c_i - c_o)\ \&\ G\ h_h\ (c_i - c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0\ DINV\ (dI\ h_l\ h_h\ (c_i - c_o))$
$ELSE\ x' = f\ (-c_o)\ \&\ G\ h_l\ (-c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0\ DINV\ (dI\ h_l\ h_h\ (-c_o))$

abbreviation $tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau \equiv LOOP\ (ctrl\ h_l\ h_h\ ;\ dyn\ c_i\ c_o\ h_l\ h_h\ \tau)$ $INV\ (I\ h_l\ h_h)$

lemma *tank-inv*:
assumes $0 \le \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\{I\ h_l\ h_h\}\ tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau\ \{I\ h_l\ h_h\}$
apply(*hyb-hoare* $\mathbf{U}(I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h)$)
prefer *4* **prefer** *7* **using** *tank-diff-inv assms* **apply** *force*+
using *assms tank-inv-arith1 tank-inv-arith2* **by** *rel-auto'*

— Refined with invariants

lemma *R-tank-inv*:
assumes $0 \le \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $[I\ h_l\ h_h,\ I\ h_l\ h_h] \ge tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau$
proof−
have $[I\ h_l\ h_h,\ I\ h_l\ h_h] \ge LOOP\ ((t ::= 0);[I\ h_l\ h_h \wedge t = 0,\ I\ h_l\ h_h])\ INV\ I\ h_l\ h_h$ (**is** - $\ge$ *?R*)
by (*refinement*, *rel-auto'*)
moreover have
$?R \ge LOOP\ ((t ::= 0);(h_0 ::= h);[I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h,\ I\ h_l\ h_h])\ INV\ I\ h_l\ h_h$ (**is** - $\ge$ *?R*)
by (*refinement*, *rel-auto'*)
moreover have
$?R \ge LOOP\ (ctrl\ h_l\ h_h;[I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h,\ I\ h_l\ h_h])\ INV\ I\ h_l\ h_h$ (**is** - $\ge$ *?R*)
by (*simp only*: *mult.assoc*, *refinement*; (*force*)*?*, (*rule R-assign-law*)*?*) *rel-auto'*
moreover have
$?R \ge LOOP\ (ctrl\ h_l\ h_h;\ dyn\ c_i\ c_o\ h_l\ h_h\ \tau)\ INV\ I\ h_l\ h_h$
apply(*simp only*: *mult.assoc*, *refinement*; (*simp*)*?*)
prefer *4* **using** *tank-diff-inv assms* **apply** *force*+
using *tank-inv-arith1 tank-inv-arith2 assms* **by** *rel-auto'*
ultimately show $[I\ h_l\ h_h,\ I\ h_l\ h_h] \ge tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau$
by *auto*
qed

no-notation *ftank* (f)
and *tank-flow* (φ)

and *tank-guard* (G)
and *tank-loop-inv* (I)
and *tank-diff-inv* (dI)
and *tank-ctrl* $(ctrl)$
and *tank-dyn-dinv* (dyn)

end

4 Hybrid Programs Preliminaries

theory *utp-hyprog-prelim*
imports
UTP.utp
Ordinary-Differential-Equations.ODE-Analysis
HOL−Analysis.Analysis
HOL−Library.Function-Algebras
Dynamics.Derivative-extra
begin recall-syntax

4.1 Continuous Variable Lenses

We begin by defining some lenses that will be useful in characterising continuous variables

4.1.1 Finite Cartesian Product Lens

definition *vec-lens* :: $'i \Rightarrow ('a \Longrightarrow 'a\hat{\ }'i)$ **where**
[*lens-defs*]: *vec-lens* k = (| *lens-get* = (λ s. *vec-nth* s k), *lens-put* = (λ s v. (χ x. *fun-upd* (*vec-nth* s) k v x)) |)

lemma *vec-vwb-lens* [*simp*]: *vwb-lens* (*vec-lens* k)
apply (*unfold-locales*)
apply (*simp-all add*: *vec-lens-def fun-eq-iff*)
using *vec-lambda-unique* **apply** *force*
done

4.1.2 Executable Euclidean Space Lens

abbreviation *eucl-nth* $k \equiv (\lambda\ x.$ *list-of-eucl* x ! $k)$

lemma *bounded-linear-eucl-nth*:
$k <$ *DIM*('a::*executable-euclidean-space*) $\Longrightarrow$ *bounded-linear* (*eucl-nth* k :: $'a \Rightarrow$ *real*)
by (*simp add*: *bounded-linear-inner-left*)

lemmas *has-derivative-eucl-nth* = *bounded-linear.has-derivative*[*OF bounded-linear-eucl-nth*]

lemma *has-derivative-eucl-nth-triv*:

$k < DIM('a::executable\text{-}euclidean\text{-}space) \Longrightarrow ((eucl\text{-}nth\ k :: 'a \Rightarrow real)\ has\text{-}derivative\ eucl\text{-}nth\ k)\ F$
using *bounded-linear-eucl-nth bounded-linear-imp-has-derivative* **by** *blast*

lemma *frechet-derivative-eucl-nth*:
$k < DIM('a::executable\text{-}euclidean\text{-}space) \Longrightarrow \partial(eucl\text{-}nth\ k :: 'a \Rightarrow real)\ (at\ t) = eucl\text{-}nth\ k$
by (*metis* (*full-types*) *frechet-derivative-at has-derivative-eucl-nth-triv*)

The Euclidean lens extracts the nth component of a Euclidean space

definition $eucl\text{-}lens :: nat \Rightarrow (real \Longrightarrow 'a::executable\text{-}euclidean\text{-}space)$ $(\Pi[\text{-}])$ **where**
$[lens\text{-}defs]$: $eucl\text{-}lens\ k = (\!|\ lens\text{-}get = eucl\text{-}nth\ k$
$, lens\text{-}put = (\lambda\ s\ v.\ eucl\text{-}of\text{-}list(list\text{-}update\ (list\text{-}of\text{-}eucl\ s)\ k\ v))\ |\!)$

lemma *eucl-vwb-lens* [*simp*]:
$k < DIM('a::executable\text{-}euclidean\text{-}space) \Longrightarrow vwb\text{-}lens\ (\Pi[k] :: real \Longrightarrow 'a)$
apply (*unfold-locales*)
apply (*simp-all add*: *lens-defs eucl-of-list-inner*)
apply (*metis eucl-of-list-list-of-eucl list-of-eucl-nth list-update-id*)
done

lemma *eucl-lens-indep* [*simp*]:
$[\![\ i < DIM('a); j < DIM('a); i \neq j\ ]\!] \Longrightarrow (eucl\text{-}lens\ i :: real \Longrightarrow 'a::executable\text{-}euclidean\text{-}space) \bowtie eucl\text{-}lens\ j$
by (*unfold-locales*, *simp-all add*: *lens-defs list-update-swap eucl-of-list-inner*)

lemma *bounded-linear-eucl-get* [*simp*]:
$k < DIM('a::executable\text{-}euclidean\text{-}space) \Longrightarrow bounded\text{-}linear\ (get_{\Pi[k]\ ::\ real\ \Longrightarrow\ 'a})$
by (*metis bounded-linear-eucl-nth eucl-lens-def lens.simps*(*1*))

Characterising lenses that are equivalent to Euclidean lenses

definition $is\text{-}eucl\text{-}lens :: (real \Longrightarrow 'a::executable\text{-}euclidean\text{-}space) \Rightarrow bool$ **where**
$is\text{-}eucl\text{-}lens\ x = (\exists\ k.\ k < DIM('a) \wedge x \approx_L \Pi[k])$

lemma *eucl-lens-is-eucl*:
$k < DIM('a::executable\text{-}euclidean\text{-}space) \Longrightarrow is\text{-}eucl\text{-}lens\ (\Pi[k] :: real \Longrightarrow 'a)$
by (*force simp add*: *is-eucl-lens-def*)

lemma *eucl-lens-is-vwb* [*simp*]: $is\text{-}eucl\text{-}lens\ x \Longrightarrow vwb\text{-}lens\ x$
using *eucl-vwb-lens is-eucl-lens-def lens-equiv-def sublens-pres-vwb* **by** *blast*

lemma *bounded-linear-eucl-lens*: $is\text{-}eucl\text{-}lens\ x \Longrightarrow bounded\text{-}linear\ (get_x)$
oops

4.2 Hybrid state space

A hybrid state-space consists, minimally, of a suitable topological space that occupies the continuous variables. Usually, $'c$ will be a Euclidean space or real vector.

alphabet *'c::t2-space hybs* =
  cvec :: *'c*

The remainder of the state-space is discrete and we make no requirements of it

abbreviation *dst* ≡ *hybs.more*$_L$

notation *cvec* (**c**)
notation *dst* (**d**)

We define hybrid expressions, predicates, and relations (i.e. programs) by utilising the hybrid state-space type.

type-synonym (*'a*, *'c*, *'s*) *hyexpr* = (*'a*, (*'c*, *'s*) *hybs-scheme*) *uexpr*
type-synonym (*'c*, *'s*) *hypred* = (*'c*, *'s*) *hybs-scheme upred*
type-synonym (*'c*, *'s*) *hyrel* = (*'c*, *'s*) *hybs-scheme hrel*

4.3 Syntax

syntax
  -eucl-lens :: *logic* ⇒ *svid* (Π[-])
  -cvec-lens :: *svid* (**c**)
  -dst-lens :: *svid* (**d**)

translations
  -eucl-lens x == *CONST eucl-lens x*
  -cvec-lens == *CONST cvec*
  -dst-lens == *CONST dst*

end

5 Derivatives of UTP Expressions

theory *utp-hyprog-deriv*
  imports *utp-hyprog-prelim*
begin

syntax
  -uscaleR :: *logic* ⇒ *logic* ⇒ *logic* (**infixr** $*_R$ *75*)
  -unorm :: *logic* ⇒ *logic* (‖-‖)

translations
  n $*_R$ *x* => *CONST bop CONST scaleR n x*
  ‖*x*‖ => *CONST uop CONST norm x*

We provide functions for specifying differentiability and taking derivatives of UTP expressions. The expressions have a hybrid state space, and so we only require differentiability of the continuous variable vector. The remainder of the state space is left unchanged by differentiation.

5.1 Differentiability

lift-definition *uexpr-differentiable* ::
$('a{::}ordered\text{-}euclidean\text{-}space, 'c{::}ordered\text{-}euclidean\text{-}space, 's)\ hyexpr \Rightarrow bool\ (differentiable_e)$
is $\lambda\ f.\ \forall\ s.\ (\lambda\ x.\ f\ (put_{cvec}\ s\ x))\ differentiable\ (at\ (get_{cvec}\ s))$ **.**

declare *uexpr-differentiable-def* [*upred-defs*]

update-uexpr-rep-eq-thms

lemma *udifferentiable-consts* [*closure*]:
$differentiable_e\ 0\ differentiable_e\ 1\ differentiable_e\ (numeral\ n)\ differentiable_e\ \ll k \gg$
by (*rel-simp*)+

lemma *udifferentiable-var* [*closure*]:
$k < DIM('c{::}executable\text{-}euclidean\text{-}space) \Longrightarrow differentiable_e(var\ ((eucl\text{-}lens\ k ::$
$real \Longrightarrow 'c)\ ;_L\ cvec))$
by (*rel-simp*)

lemma *udifferentiable-pr-var* [*closure*]:
$k < DIM('c{::}executable\text{-}euclidean\text{-}space) \Longrightarrow differentiable_e\ (var\ (pr\text{-}var\ ((eucl\text{-}lens$
$k :: real \Longrightarrow 'c)\ ;_L\ cvec)))$
by (*rel-simp*)

lemma *udifferentiable-plus* [*closure*]:
$[\![\ differentiable_e\ e;\ differentiable_e\ f\ ]\!] \Longrightarrow differentiable_e\ (e + f)$
by (*rel-simp*)

lemma *udifferentiable-uminus* [*closure*]:
$[\![\ differentiable_e\ e\ ]\!] \Longrightarrow differentiable_e\ (-\ e)$
by (*rel-simp*)

lemma *udifferentiable-minus* [*closure*]:
$[\![\ differentiable_e\ e;\ differentiable_e\ f\ ]\!] \Longrightarrow differentiable_e\ (e - f)$
by (*rel-simp*)

lemma *udifferentiable-mult* [*closure*]:
fixes $e\ f :: ('a{::}\{ordered\text{-}euclidean\text{-}space, real\text{-}normed\text{-}algebra\}, 'c{::}ordered\text{-}euclidean\text{-}space,$
$'s)\ hyexpr$
shows $[\![\ differentiable_e\ e;\ differentiable_e\ f\ ]\!] \Longrightarrow differentiable_e\ (e * f)$
by (*rel-simp*)

lemma *udifferentiable-scaleR* [*closure*]:
fixes $e :: ('a{::}ordered\text{-}euclidean\text{-}space, 'c{::}ordered\text{-}euclidean\text{-}space, 's)\ hyexpr$
shows $[\![\ differentiable_e\ n;\ differentiable_e\ e\ ]\!] \Longrightarrow differentiable_e\ \mathbf{U}(n *_R e)$
by (*rel-simp*)

lemma *udifferentiable-power* [*closure*]:
fixes e :: ($'a$::{*ordered-euclidean-space*, *real-normed-field*}, $'c$::*ordered-euclidean-space*, $'s$) *hyexpr*
shows *differentiable*$_e$ $e \Longrightarrow$ *differentiable*$_e$ (e ^ n)
by (*rel-simp*)

lemma *udifferentiable-norm* [*closure*]:
fixes e :: ($'a$::*ordered-euclidean-space*, $'c$::*ordered-euclidean-space*, $'s$) *hyexpr*
shows $\llbracket$ *differentiable*$_e$ e; $\bigwedge$ s. $e\llbracket \ll s \gg /\&\mathbf{v} \rrbracket \neq 0$ $\rrbracket \Longrightarrow$ *differentiable*$_e$ $\mathbf{U}$(*norm* e)
by (*rel-simp*, *metis differentiable-compose differentiable-norm-at*)

5.2 Differentiation

For convenience in the use of ODEs, we differentiate with respect to a known context of derivative for the variables. This means we don't have to deal with symbolic variable derivatives and so the state space is unchanged by differentiation.

lift-definition *uexpr-deriv* ::
$'c$ *usubst* $\Rightarrow$ ($'a$::*ordered-euclidean-space*, $'c$::*ordered-euclidean-space*, $'s$) *hyexpr* $\Rightarrow$ ($'a$, $'c$, $'s$) *hyexpr* ((- $\vdash \partial_e$ -) [*100*, *101*] *100*)
is λ σ f s. *frechet-derivative* (λ x. f (put_{cvec} s x)) (*at* (get_{cvec} s)) (σ (get_{cvec} s))
.

declare *uexpr-deriv-def* [*upred-defs*]

update-uexpr-rep-eq-thms

no-utp-lift *uexpr-deriv*

named-theorems *uderiv*

lemma *uderiv-zero* [*uderiv*]: $F' \vdash \partial_e$ *0* = *0*
by (*rel-simp*, *simp add*: *frechet-derivative-const*)

lemma *uderiv-one* [*uderiv*]: $F' \vdash \partial_e$ *1* = *0*
by (*rel-simp*, *simp add*: *frechet-derivative-const*)

lemma *uderiv-numeral* [*uderiv*]: $F' \vdash \partial_e$ (*numeral* n) = *0*
by (*rel-simp*, *simp add*: *frechet-derivative-const*)

lemma *uderiv-lit* [*uderiv*]: $F' \vdash \partial_e$ ($\ll v \gg$) = *0*
by (*rel-simp*, *simp add*: *frechet-derivative-const*)

lemma *uderiv-plus* [*uderiv*]:
$\llbracket$ *differentiable*$_e$ e; *differentiable*$_e$ f $\rrbracket \Longrightarrow F' \vdash \partial_e (e + f) = (F' \vdash \partial_e\ e + F' \vdash \partial_e\ f)$
by (*rel-simp*, *simp add*: *frechet-derivative-plus*)

lemma *uderiv-uminus* [*uderiv*]:
differentiable$_e$ *e* $\Longrightarrow$ $F' \vdash \partial_e$ (− *e*) = − ($F' \vdash \partial_e$ *e*)
by (*rel-simp*, *simp add*: *frechet-derivative-uminus*)

lemma *uderiv-minus* [*uderiv*]:
⟦ *differentiable*$_e$ *e*; *differentiable*$_e$ *f* ⟧ $\Longrightarrow$ $F' \vdash \partial_e$ (*e* − *f*) = ($F' \vdash \partial_e$ *e*) − (F' $\vdash \partial_e$ *f*)
by (*rel-simp*, *simp add*: *frechet-derivative-minus*)

lemma *uderiv-mult* [*uderiv*]:
fixes *e f* :: ('*a*::{*ordered-euclidean-space*, *real-normed-algebra*}, '*c*::*ordered-euclidean-space*, '*s*) *hyexpr*
shows ⟦ *differentiable*$_e$ *e*; *differentiable*$_e$ *f* ⟧ $\Longrightarrow$ $F' \vdash \partial_e$ (*e* ∗ *f*) = (*e* ∗ $F' \vdash$ ∂_e *f* + $F' \vdash \partial_e$ *e* ∗ *f*)
by (*rel-simp*, *simp add*: *frechet-derivative-mult*)

lemma *uderiv-scaleR* [*uderiv*]:
fixes *f* :: ('*a*::{*ordered-euclidean-space*, *real-normed-algebra*}, '*c*::*ordered-euclidean-space*, '*s*) *hyexpr*
shows ⟦ *differentiable*$_e$ *e*; *differentiable*$_e$ *f* ⟧ $\Longrightarrow$ $F' \vdash \partial_e$ **U**(*e* $*_R$ *f*) = **U**(*e* $*_R$ $F' \vdash \partial_e$ *f* + $F' \vdash \partial_e$ *e* $*_R$ *f*)
by (*rel-simp*, *simp add*: *frechet-derivative-scaleR*)

lemma *uderiv-power* [*uderiv*]:
fixes *e* :: ('*a*::{*ordered-euclidean-space*, *real-normed-field*}, '*c*::*ordered-euclidean-space*, '*s*) *hyexpr*
shows *differentiable*$_e$ *e* $\Longrightarrow$ $F' \vdash \partial_e$ (*e* ˆ *n*) = *of-nat n* ∗ $F' \vdash \partial_e$ *e* ∗ *e* ˆ (*n* − *1*)
by (*rel-simp*, *simp add*: *frechet-derivative-power ueval*)

The derivative of a variable represented by a Euclidean lens into the continuous state space uses the said lens to obtain the derivative from the context F'

lemma *uderiv-var*:
fixes F' :: '*c*::*executable-euclidean-space usubst*
assumes *k* < *DIM*('*c*)
shows $F' \vdash \partial_e$ (*var* ((Π[*k*] :: *real* $\Longrightarrow$ '*c*) ;$_L$ **c**)) = ⟨F'⟩$_s$ Π[*k*] $\oplus_p$ *cvec*
using *assms*
by (*rel-simp*, *metis bounded-linear-imp-has-derivative bounded-linear-inner-left frechet-derivative-at*)

lemma *uderiv-pr-var* [*uderiv*]:
fixes F' :: '*c*::*executable-euclidean-space usubst*
assumes *k* < *DIM*('*c*)
shows $F' \vdash \partial_e$ &**c**:Π[*k*] = ⟨F'⟩$_s$ Π[*k*] $\oplus_p$ **c**
using *assms* **by** (*simp add*: *pr-var-def uderiv-var*)

end

5.3 Examples

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

theory *KAT-rKAT-exuclid-Examples-ndfun*
 imports *KAT-rKAT-rVCs-ndfun utp-hyprog-deriv*

begin

declare [[*coercion Rep-uexpr*]]

— Frechet derivatives

no-notation *dual* (∂)
 and *n-op* (*n* - [*90*] *91*)
 and *vec-nth* (**infixl** $ *90*)

notation *vec-nth* (**infixl** ¡ *90*)

abbreviation e *k* ≡ *axis k* (*1*::*real*)

lemma *frechet-derivative-id*:
 fixes *t*::*'a* :: {*inverse*,*banach*,*real-normed-algebra-1*}
 shows ∂ (λ*t*::*'a*. *t*) (*at t*) = (λ*t*. *t*)
 using *frechet-derivative-at*[*OF has-derivative-id*] **unfolding** *id-def* **..**

lemma *has-derivative-exp*: *D exp* ↦ (λ*t*. *t* · *exp x*) *at x within T* **for** *x*::*real*
 by (*auto intro*!: *derivative-intros*)

lemma *has-derivative-exp-compose*:
 fixes *f*::*real* ⇒ *real*
 assumes *D f* ↦ *f'* *at y within T*
 shows *D* (λ*t*. *exp* (*f t*)) ↦ (λ*x*. *f'* *x* · *exp* (*f y*)) *at y within T*
 using *has-derivative-compose*[*OF assms has-derivative-exp*] **by** *simp*

lemma *frechet-derivative-works1*: *f differentiable* (*at t*) ⟹ (*D f* ↦ (∂ *f* (*at t*)) (*at t*)) **for** *t*::*real*
 by (*simp add*: *frechet-derivative-works*)

lemmas *frechet-derivative-exp* =
 frechet-derivative-works1[*THEN frechet-derivative-at*[*OF has-derivative-exp-compose*, *symmetric*]]

lemma *differentiable-exp*[*simp*]: *exp differentiable* (*at x*) **for** *x*::*'a*::{*banach*,*real-normed-field*}
 unfolding *differentiable-def* **using** *DERIV-exp*[*of x*] **unfolding** *has-field-derivative-def*
by *blast*

lemma *differentiable-sin*[*simp*]: *sin differentiable* (*at x*) **for** *x*::*'a*::{*banach*,*real-normed-field*}
 unfolding *differentiable-def* **using** *DERIV-sin*[*of x*] **unfolding** *has-field-derivative-def*

by *blast*

lemma *differentiable-cos*[*simp*]: *cos differentiable* (*at x*) **for** *x*::′*a*::{*banach*,*real-normed-field*}
unfolding *differentiable-def* **using** *DERIV-cos*[*of x*] **unfolding** *has-field-derivative-def*
by *blast*

lemma *differentiable-exp-compose*[*derivative-intros*]:
fixes *f*::′*a*::*real-normed-vector* ⇒ ′*b*::{*banach*,*real-normed-field*}
shows *f differentiable* (*at x*) ⟹ (λ*t*. *exp* (*f t*)) *differentiable* (*at x*)
by (*rule differentiable-compose*[*of exp*], *simp-all*)

named-theorems *frechet-simps simplification rules for Frechet derivatives*

declare *frechet-derivative-plus* [*frechet-simps*]
frechet-derivative-minus [*frechet-simps*]
frechet-derivative-uminus [*frechet-simps*]
frechet-derivative-mult [*frechet-simps*]
frechet-derivative-power [*frechet-simps*]
frechet-derivative-exp [*frechet-simps*]
frechet-derivative-sin [*frechet-simps*]
frechet-derivative-id [*frechet-simps*]
frechet-derivative-const [*frechet-simps*]

method *frechet-derivate*
= (*subst frechet-simps*; (*frechet-derivate*)?)

lemma $D\ (\lambda t.\ a * t^2 + v * t + x) = (\lambda t.\ 2 * a * t + v)$ *on T*
by(*auto intro*!: *poly-derivatives*)

lemma $\partial\ (\lambda t.\ a \cdot t^2 + v \cdot t + x)\ (at\ t) = (\lambda x.\ x \cdot (2 \cdot a \cdot t + v))$ **for** *t*::*real*
by (*simp add*: *frechet-simps field-simps*)

lemma $D\ (\lambda t.\ a5 * t\hat{}5 - a2 * exp\ (t\hat{}2) + a1 * sin\ t + a0) =$
$(\lambda t.\ 5 * a5 * t\hat{}4 - 2 * a2 * t * exp\ (t\hat{}2) + a1 * cos\ t)$ *on T*
by (*auto intro*!: *poly-derivatives*)

lemma $\partial\ (\lambda t.\ a5 \cdot t\ \hat{}\ 5 - a2 \cdot exp\ (t^2) + a1 \cdot sin\ t + a0)\ (at\ t) =$
$(\lambda x.\ x \cdot (5 \cdot a5 \cdot t\ \hat{}\ 4 - 2 \cdot a2 \cdot t \cdot exp\ (t^2) + a1 \cdot cos\ t))$ **for** *t*::*real*
by (*frechet-derivate*, *auto simp*: *field-simps intro*!: *derivative-intros*)

utp-lit-vars

— A tactic for verification of hybrid programs

named-theorems *hoare-intros*

declare *H-assign-init* [*hoare-intros*]
and *H-cond* [*hoare-intros*]
and *local-flow.H-g-ode-ivl* [*hoare-intros*]

and *H-g-ode-inv* [*hoare-intros*]

method *body-hoare*
= (*rule hoare-intros*,(*simp*)?; *body-hoare?*)

method *hyb-hoare* **for** *P*::′*a upred*
= (*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-loop-law* [*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*)?)

declare *eucl-of-list-def* [*simp*]
and *axis-def* [*simp*]

— Preliminary lemmas for type 2

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

declare *forall-2* [*simp*]

instance *integer* :: *order-lean*
by *intro-classes auto*

lemma *enum-2*[*simp*]: (*enum-class.enum*::*2 list*) = [*0*::*2*, *1*]
by *code-simp+*

lemma *basis-list2*[*simp*]: *Basis-list* = [e (*0*::*2*), e *1*]
by (*auto simp*: *Basis-list-vec-def Basis-list-real-def*)

lemma *list-of-eucl2*[*simp*]: *list-of-eucl* (*s*::*real^2*) = *map* ((·) *s*) [e (*0*::*2*), e *1*]
unfolding *list-of-eucl-def* **by** *simp*

lemma *inner-axis2*[*simp*]: $x \cdot (\chi\ j{::}2.\ \mathit{if}\ j = i\ \mathit{then}\ (k{::}\mathit{real})\ \mathit{else}\ 0) = (x¡i) \cdot k$
unfolding *inner-vec-def UNIV-2 inner-real-def* **using** *exhaust-2* **by** *force*

— Preliminary lemmas for type 2

declare *forall-4* [*simp*]

lemma *four-eq-zero*[*simp*]: $(4{::}4) = 0$
by *simp*

lemma *enum-4*[*simp*]: (*enum-class.enum*::*4 list*) = [*0*::*4*, *1*, *2*, *3*]
by *code-simp+*

lemma *basis-list4*[*simp*]: *Basis-list* = [e (*0*::*4*), e *1*, e *2*, e *3*]
by (*auto simp*: *Basis-list-vec-def Basis-list-real-def*)

lemma *list-of-eucl4*[*simp*]: *list-of-eucl* (*s*::*real*ˆ*4*) = *map* ((·) *s*) [e (*0*::*4*), e *1*, e *2*, e *3*]
unfolding *list-of-eucl-def* **by** *simp*

lemma *inner-axis4*[*simp*]: $x \cdot (\chi\ j{::}4.\ \mathit{if}\ j = i\ \mathit{then}\ (k{::}\mathit{real})\ \mathit{else}\ 0) = (x¡i) \cdot k$
unfolding *inner-vec-def UNIV-4 inner-real-def* **using** *exhaust-4* **by** *force*

5.3.1 Pendulum

abbreviation x :: *real* $\Longrightarrow$ *real*ˆ*2* **where** $x \equiv \Pi[0]$
abbreviation y :: *real* $\Longrightarrow$ *real*ˆ*2* **where** $y \equiv \Pi[\mathit{Suc}\ 0]$

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 prove that this motion remains circular.

abbreviation *fpend* :: (*real*ˆ*2*) *usubst* (f)
where *fpend* $\equiv [x \mapsto_s y,\ y \mapsto_s -x]$

abbreviation *pend-flow* :: *real* ⇒ (*real*ˆ*2*) *usubst* (φ)
where *pend-flow* $\tau \equiv [x \mapsto_s x \cdot \cos \tau + y \cdot \sin \tau,\ y \mapsto_s - x \cdot \sin \tau + y \cdot \cos \tau]$

— Verified with annotated dynamics

lemma *pendulum-dyn*: $\{r^2 = x^2 + y^2\}(\mathit{EVOL}\ \varphi\ G\ T)\{r^2 = x^2 + y^2\}$
by (*simp*, *pred-simp*)

— Verified with invariants

lemma *pendulum-inv*: $\{r^2 = x^2 + y^2\}\ (x\acute{}= f\ \&\ G)\ \{r^2 = x^2 + y^2\}$
by (*pred-simp*, *auto intro*!: *diff-invariant-rules poly-derivatives*)

— Verified by providing solutions

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

lemma *pendulum-flow*: $\{r^2 = x^2 + y^2\}$ $(x\acute{}= f \ \& \ G)$ $\{r^2 = x^2 + y^2\}$
by (*simp only*: *local-flow.sH-g-ode*[*OF local-flow-pend*], *pred-simp*)

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

5.3.2 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 assigntment that flips the velocity, thus it is a completely elastic collision with the ground. We prove that the ball remains above ground and below its initial resting position.

abbreviation v :: *real* $\Longrightarrow$ *real*ˆ*2*
where $v \equiv \Pi[Suc\ 0]$

abbreviation *fball* :: *real* $\Rightarrow$ (*real*, *2*) *vec* $\Rightarrow$ (*real*, *2*) *vec* (f)
where $f\ g \equiv [x \mapsto_s v,\ v \mapsto_s g]$

abbreviation *ball-flow* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*ˆ*2*) *usubst* (φ)
where $\varphi\ g\ \tau \equiv [x \mapsto_s g \cdot \tau\ \hat{}\ 2/2 + v \cdot \tau + x,\ \ v \mapsto_s g \cdot \tau + v]$

— Verified with invariants

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

lemma [*bb-real-arith*]:
fixes $x\ v$ :: *real*
assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
shows $(x{::}real) \leq h$
proof–
have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
using *inv* **and** ‹$0 > g$› **by** *auto*
hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
hence $(v \cdot v)/(2 \cdot g) = (x - h)$
by *auto*
also from *obs* **have** $(v \cdot v)/(2 \cdot g) \leq 0$
using *divide-nonneg-neg* **by** *fastforce*

ultimately have $h - x \geq 0$
  by *linarith*
thus *?thesis* **by** *auto*
qed

lemma *fball-invariant*:
  fixes *g h* :: *real*
  defines *dinv*: $I \equiv \mathbf{U}(2 \cdot «g» \cdot x - 2 \cdot «g» \cdot «h» - (v \cdot v) = 0)$
  shows *diff-invariant I (f g) UNIV UNIV 0 G*
  unfolding *dinv* **apply**(*pred-simp*, *rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
  by (*auto intro*!: *poly-derivatives*)

abbreviation *bb-dinv g h* ≡
  (*LOOP*
    $((x' = f\ g\ \&\ (x \geq 0)\ DINV\ (2 \cdot g \cdot x - 2 \cdot g \cdot h - v \cdot v = 0));$
    $(IF\ (v = 0)\ THEN\ (v ::= -v)\ ELSE\ skip))$
  $INV\ (0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v))$

lemma *bouncing-ball-inv*: $g < 0 \Longrightarrow h \geq 0 \Longrightarrow \{x = h \wedge v = 0\}$ *bb-dinv g h* $\{0 \leq x \wedge x \leq h\}$
  apply(*hyb-hoare* $\mathbf{U}(0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v))$
  using *fball-invariant* **apply** (*simp-all*)
  by (*rel-auto′ simp*: *bb-real-arith*)

— Verified with annotated dynamics

lemma [*bb-real-arith*]:
  fixes *x v* :: *real*
  assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
    and *pos*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$
  shows $2 \cdot g \cdot h + (-(g \cdot \tau) - v) \cdot (-(g \cdot \tau) - v) = 0$
    and $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
  from *pos* **have** $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
  then have $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$
    by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right monoid-mult-class.power2-eq-square semiring-class.distrib-left*)
  hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot h = 0$
    using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
  hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
   apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

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

```
    by (simp add: monoid-mult-class.power2-eq-square)
qed

lemma [bb-real-arith]:
  fixes x v :: real
  assumes invar: 2 · g · x = 2 · g · h + v · v
  shows 2 · g · (g · τ² / 2 + v · τ + (x::real)) =
  2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) (is ?lhs = ?rhs)
proof–
  have ?lhs = g² · τ² + 2 · g · v · τ + 2 · g · x
      apply(subst Rat.sign-simps(18))+
      by(auto simp: semiring-normalization-rules(29))
    also have ... = g² · τ² + 2 · g · v · τ + 2 · g · h + v · v (is ... = ?middle)
      by(subst invar, simp)
    finally have ?lhs = ?middle.
  moreover
  {have ?rhs = g · g · (τ · τ) + 2 · g · v · τ + 2 · g · h + v · v
    by (simp add: Groups.mult-ac(2,3) semiring-class.distrib-left)
  also have ... = ?middle
    by (simp add: semiring-normalization-rules(29))
  finally have ?rhs = ?middle.}
  ultimately show ?thesis by auto
qed

abbreviation bb-evol g h T ≡
  (LOOP (
    (EVOL (φ g) (x ≥ 0) T);
    (IF (v = 0) THEN (v ::= −v) ELSE skip))
  INV (0 ≤ x ∧ 2 · g · x = 2 · g · h + v · v))

lemma bouncing-ball-dyn:
  assumes g < 0 and h ≥ 0
  shows {x = h ∧ v = 0} bb-evol g h T {0 ≤ x ∧ x ≤ h}
  apply(hyb-hoare U(0 ≤ x ∧ 2 · g · x = 2 · g · h + v · v))
  using assms by (rel-auto' simp: bb-real-arith)

— Verified by providing solutions

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

abbreviation bb-sol g h ≡
  (LOOP (
    (x´= f g & (x ≥ 0));
    (IF (v = 0) THEN (v ::= −v) ELSE skip))
```

INV $(0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v))$

lemma *bouncing-ball-flow*:
assumes $g < 0$ **and** $h \geq 0$
shows $\{x = h \wedge v = 0\}$ *bb-sol g h* $\{0 \leq x \wedge x \leq h\}$
apply(*hyb-hoare* $\mathbf{U}(0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v)$)
apply(*subst local-flow.sH-g-ode*[*OF local-flow-ball*])
using *assms* **by** (*rel-auto′ simp*: *bb-real-arith*)

— Refined with annotated dynamics

lemma *R-bb-assign*: $g < (0::real) \Longrightarrow 0 \leq h \Longrightarrow$
$[v = 0 \wedge 0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v,\ 0 \leq x \wedge 2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v] \geq (v ::= -v)$
by (*rule R-assign-law*, *pred-simp*)

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows $[x = h \wedge v = 0,\ 0 \leq x \wedge x \leq h] \geq$ *bb-evol g h T*
apply(*refinement*; (*rule R-bb-assign*[*OF assms*])?)
using *assms* **by** (*rel-auto′ simp*: *bb-real-arith*)

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

5.3.3 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 $T' = -a * (T - c)$ where $c = L \geq 0$ when the heater is on, and $c = 0$ when it is off. We prove that the thermostat keeps the room's temperature between T_l and T_h.

hide-const t

abbreviation T :: *real* $\Longrightarrow$ *real^4* **where** $T \equiv \Pi[0]$
abbreviation t :: *real* $\Longrightarrow$ *real^4* **where** $t \equiv \Pi[1]$
abbreviation T_0 :: *real* $\Longrightarrow$ *real^4* **where** $T_0 \equiv \Pi[2]$
abbreviation ϑ :: *real* $\Longrightarrow$ *real^4* **where** $\vartheta \equiv \Pi[3]$

abbreviation *ftherm* :: *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*, *4*) *vec* $\Rightarrow$ (*real*, *4*) *vec* (f)
where $f\ a\ c \equiv [T \mapsto_s -(a * (T - c)),\ T_0 \mapsto_s 0,\ \vartheta \mapsto_s 0,\ t \mapsto_s 1]$

abbreviation *therm-guard* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real^4*) *upred* (G)
where $G\ T_l\ T_h\ a\ L \equiv \mathbf{U}(t \leq -(ln\ ((L-(if\ L{=}0\ then\ T_l\ else\ T_h))/(L-T_0)))/a)$

no-utp-lift *therm-guard* (*0 1 2 3*)

abbreviation *therm-loop-inv* :: *real* ⇒ *real* ⇒ (*real^4*) *upred* (*I*)
  where $I\ T_l\ T_h \equiv \mathbf{U}(T_l \le T \wedge T \le T_h \wedge (\vartheta = 0 \vee \vartheta = 1))$

no-utp-lift *therm-loop-inv* (*0 1*)

abbreviation *therm-flow* :: *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *usubst* (φ)
  where $\varphi\ a\ c\ \tau \equiv [T \mapsto_s -\ exp(-a * \tau) * (c - T) + c,\ t \mapsto_s \tau + t,\ T_0 \mapsto_s T_0,\ \vartheta \mapsto_s \vartheta]$

abbreviation *therm-ctrl* :: *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (*ctrl*)
  where $ctrl\ T_l\ T_h \equiv$
  $(t ::= 0); (T_0 ::= T);$
  $(IF\ (\vartheta = 0 \wedge T_0 \le T_l + 1)\ THEN\ (\vartheta ::= 1)\ ELSE$
   $IF\ (\vartheta = 1 \wedge T_0 \ge T_h - 1)\ THEN\ (\vartheta ::= 0)\ ELSE\ skip)$

abbreviation *therm-dyn* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real^4*) *nd-fun* (*dyn*)
  where $dyn\ T_l\ T_h\ a\ T_u\ \tau \equiv$
  $IF\ (\vartheta = 0)\ THEN\ x' = f\ a\ 0\ \&\ G\ T_l\ T_h\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0$
   $ELSE\ x' = f\ a\ T_u\ \&\ G\ T_l\ T_h\ a\ T_u\ on\ \{0..\tau\}\ UNIV\ @\ 0$

abbreviation $therm\ T_l\ T_h\ a\ L\ \tau \equiv LOOP\ (ctrl\ T_l\ T_h\ ;\ dyn\ T_l\ T_h\ a\ L\ \tau)\ INV\ (I\ T_l\ T_h)$

— Verified by providing solutions

lemma *norm-diff-therm-dyn*: $0 < (a::real) \Longrightarrow (a \cdot (s_2¡0 - T_u) - a \cdot (s_1¡0 - T_u))^2$
  $\le (a \cdot sqrt\ ((s_1¡1 - s_2¡1)^2 + ((s_1¡2 - s_2¡2)^2 + ((s_1¡3 - s_2¡3)^2 + (s_1¡0 - s_2¡0)^2))))^2$
proof(*simp add*: *field-simps*)
  assume *a1*: $0 < a$
  have $(a \cdot s_2¡0 - a \cdot s_1¡0)^2 = a^2 \cdot (s_2¡0 - s_1¡0)^2$
    by (*metis* (*mono-tags*, *hide-lams*) *Rings.ring-distribs*(*4*) *mult.left-commute* *semiring-normalization-rules*(*18*) *semiring-normalization-rules*(*29*))
  moreover have $(s_2¡0 - s_1¡0)^2 \le (s_1¡0 - s_2¡0)^2 + ((s_1¡1 - s_2¡1)^2 + ((s_1¡2 - s_2¡2)^2 + (s_1¡3 - s_2¡3)^2))$
    using *zero-le-power2* **by** (*simp add*: *power2-commute*)
  thus $(a \cdot s_2¡0 - a \cdot s_1¡0)^2 \le a^2 \cdot (s_1¡1 - s_2¡1)^2 +$
  $(a^2 \cdot (s_1¡0 - s_2¡0)^2 + (a^2 \cdot (s_1¡2 - s_2¡2)^2 + a^2 \cdot (s_1¡3 - s_2¡3)^2))$
    using *a1* **by** (*simp add*: *Groups.algebra-simps*(*18*)[*symmetric*] *calculation*)
qed

lemma *local-lipschitz-therm-dyn*:
  assumes $0 < (a::real)$
  shows *local-lipschitz UNIV UNIV* ($\lambda t::real.\ f\ a\ T_u$)
  apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)
  apply(*clarsimp*, *rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=a* **in** *exI*)
 using *assms* **apply**(*simp add*: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *pred-simp*)

unfolding *real-sqrt-abs*[*symmetric*] **apply** (*rule real-le-lsqrt*)
by (*simp-all add*: *norm-diff-therm-dyn*)

lemma *local-flow-therm*: $a > 0 \Longrightarrow$ *local-flow* ($f\ a\ T_u$) *UNIV UNIV* ($\varphi\ a\ T_u$)
apply (*unfold-locales*, *simp-all*)
using *local-lipschitz-therm-dyn* **apply** *pred-simp*
apply(*pred-simp*, *force intro*!: *poly-derivatives*)
using *exhaust-4* **by** (*rel-auto′ simp*: *vec-eq-iff*)

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

lemma *therm-dyn-up*:
fixes T::*real*
assumes $a > 0$ **and** *Thyps*: $T_l \leq T$ $T \leq T_h$ $T_h < (T_u$::*real*)
and *thyps*: $0 \leq \tau$ $\forall\tau\in\{0..\tau\}.\ \tau \leq -\ (ln\ ((T_u - T_h)\ /\ (T_u - T))\ /\ a)$
shows $T_u - T_h \leq exp\ (-(a * \tau)) * (T_u - T)$
and $T_u - exp\ (-(a * \tau)) * (T_u - T) \leq T_h$
and $T_l \leq T_u - exp\ (-(a * \tau)) * (T_u - T)$
proof−
have $0 \leq \tau \wedge \tau \leq -\ (ln\ ((T_u - T_h)\ /\ (T_u - T))\ /\ a)$
using *thyps* **by** *auto*
hence $ln\ ((T_u - T_h)\ /\ (T_u - T)) \leq -\ a * \tau \wedge -\ a * \tau \leq 0$
using *assms*(*1*) *divide-le-cancel* **by** *fastforce*
also have $(T_u - T_h)\ /\ (T_u - T) > 0$
using *Thyps* **by** *auto*
ultimately have $(T_u - T_h)\ /\ (T_u - T) \leq exp\ (-a * \tau) \wedge exp\ (-a * \tau) \leq 1$
using *exp-ln exp-le-one-iff* **by** (*metis exp-less-cancel-iff not-less*)
moreover have $T_u - T > 0$
using *Thyps* **by** *auto*
ultimately have *obs*: $(T_u - T_h) \leq exp\ (-a * \tau) * (T_u - T) \wedge exp\ (-a * \tau)$

$* (T_u - T) \leq (T_u - T)$
by (*simp add*: *pos-divide-le-eq*)
thus $(T_u - T_h) \leq exp\ (-(a * \tau)) * (T_u - T)$
by *auto*
thus $T_u - exp\ (-(a * \tau)) * (T_u - T) \leq T_h$
by *auto*
show $T_l \leq T_u - exp\ (-(a * \tau)) * (T_u - T)$
using *Thyps* **and** *obs* **by** *auto*
qed

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

lemma *thermostat-flow*:
assumes $0 < a$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $\{I\ T_l\ T_h\}$ *therm* $T_l\ T_h\ a\ T_u\ \tau$ $\{I\ T_l\ T_h\}$
apply(*hyb-hoare* **U**$(I\ T_l\ T_h \wedge t{=}0 \wedge T_0 = T)$)
prefer *4* **prefer** *8* **using** *local-flow-therm assms* **apply** *force+*
using *assms therm-dyn-up therm-dyn-down* **by** *rel-auto′*

— Refined by providing solutions

lemma *R-therm-down*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $[\vartheta = 0 \wedge I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T,\ I\ T_l\ T_h] \geq$
$(x' = f\ a\ 0\ \&\ G\ T_l\ T_h\ a\ 0\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *therm-dyn-down*[*OF assms*(*1*,*3*), *of* - T_h] *assms* **by** *rel-auto′*

lemma *R-therm-up*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $[\neg\ \vartheta = 0 \wedge I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T,\ I\ T_l\ T_h] \geq$
$(x' = f\ a\ T_u\ \&\ G\ T_l\ T_h\ a\ T_u\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
apply(*rule local-flow.R-g-ode-ivl*[*OF local-flow-therm*])
using *therm-dyn-up*[*OF assms*(*1*) - - *assms*(*4*), *of* T_l] *assms* **by** *rel-auto′*

lemma *R-therm-time*: $[I\ T_l\ T_h,\ I\ T_l\ T_h \wedge t = 0] \geq (t ::= 0)$
by (*rule R-assign-law*, *pred-simp*)

lemma *R-therm-temp*: $[I\ T_l\ T_h \wedge t = 0,\ I\ T_l\ T_h \wedge t = 0 \wedge T_0 = T] \geq (T_0 ::=$
$T)$
by (*rule R-assign-law*, *pred-simp*)

lemma *R-thermostat-flow*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_l$ **and** $T_h < T_u$
shows $[I\ T_l\ T_h,\ I\ T_l\ T_h] \geq$ *therm* $T_l\ T_h\ a\ T_u\ \tau$
by (*refinement*; (*rule R-therm-time*)?, (*rule R-therm-temp*)?, (*rule R-assign-law*)?,

(*rule R-therm-up*[*OF assms*])?, (*rule R-therm-down*[*OF assms*])?) *rel-auto′*

no-notation *ftherm* (f)
  and *therm-flow* (φ)
  and *therm-guard* (G)
  and *therm-loop-inv* (I)
  and *therm-ctrl* $(ctrl)$
  and *therm-dyn* (dyn)

5.3.4 Water tank

— Variation of Hespanha and [1]

abbreviation $h :: real \Longrightarrow real\hat{}4$ **where** $h \equiv \Pi[0]$
abbreviation $h_0 :: real \Longrightarrow real\hat{}4$ **where** $h_0 \equiv \Pi[2]$
abbreviation $\pi :: real \Longrightarrow real\hat{}4$ **where** $\pi \equiv \Pi[3]$

abbreviation *ftank* $:: real \Rightarrow (real, 4)\ vec \Rightarrow (real, 4)\ vec\ (f)$
where $f\ k \equiv [\pi \mapsto_s 0,\ h \mapsto_s k,\ h_0 \mapsto_s 0,\ t \mapsto_s 1]$

abbreviation *tank-flow* $:: real \Rightarrow real \Rightarrow (real\hat{}4)\ usubst\ (\varphi)$
where $\varphi\ k\ \tau \equiv [h \mapsto_s k * \tau + h,\ t \mapsto_s \tau + t,\ h_0 \mapsto_s h_0,\ \pi \mapsto_s \pi]$

abbreviation *tank-guard* $:: real \Rightarrow real \Rightarrow (real\hat{}4)\ upred\ (G)$
where $G\ h_x\ k \equiv \mathbf{U}(t \le (h_x - h_0)/k)$

no-utp-lift *tank-guard* $(0\ 1)$

abbreviation *tank-loop-inv* $:: real \Rightarrow real \Rightarrow (real\hat{}4)\ upred\ (I)$
where $I\ h_l\ h_h \equiv \mathbf{U}(h_l \le h \wedge h \le h_h \wedge (\pi = 0 \vee \pi = 1))$

no-utp-lift *tank-loop-inv* $(0\ 1)$

abbreviation *tank-diff-inv* $:: real \Rightarrow real \Rightarrow real \Rightarrow (real\hat{}4)\ upred\ (dI)$
where $dI\ h_l\ h_h\ k \equiv \mathbf{U}(h = k \cdot t + h_0 \wedge 0 \le t \wedge h_l \le h_0 \wedge h_0 \le h_h \wedge (\pi = 0 \vee \pi = 1))$

no-utp-lift *tank-diff-inv* $(0\ 1\ 2)$

— Verified by providing solutions

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

lemma *tank-arith*:
fixes $y::real$

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

abbreviation *tank-ctrl* :: $real \Rightarrow real \Rightarrow (real\hat{\ }4)$ *nd-fun* (*ctrl*)
where $ctrl\ h_l\ h_h \equiv (t ::= 0);(h_0 ::= h);$
$(IF\ (\pi = 0 \wedge h_0 \leq h_l + 1)\ THEN\ (\pi ::= 1)\ ELSE$
$(IF\ (\pi = 1 \wedge h_0 \geq h_h - 1)\ THEN\ (\pi ::= 0)\ ELSE\ skip))$

abbreviation *tank-dyn-sol* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow (real\hat{\ }4)$ *nd-fun* (*dyn*)
where $dyn\ c_i\ c_o\ h_l\ h_h\ \tau \equiv (IF\ (\pi = 0)\ THEN$
$(x' = f\ (c_i - c_o)\ \&\ G\ h_h\ (c_i - c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0)$
$ELSE\ (x' = f\ (-c_o)\ \&\ G\ h_l\ (-c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0))$

abbreviation $tank\text{-}sol\ c_i\ c_o\ h_l\ h_h\ \tau \equiv LOOP\ (ctrl\ h_l\ h_h\ ;\ dyn\ c_i\ c_o\ h_l\ h_h\ \tau)\ INV\ (I\ h_l\ h_h)$

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 $\{I\ h_l\ h_h\}$ $tank\text{-}sol\ c_i\ c_o\ h_l\ h_h\ \tau$ $\{I\ h_l\ h_h\}$
apply(*hyb-hoare* $\mathbf{U}(I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h)$)
prefer *4* **prefer** *8* **using** *assms local-flow-tank* **apply** *force*+
using *assms tank-arith* **by** *rel-auto'*

no-notation *tank-dyn-sol* (*dyn*)

— Verified with invariants

lemma *tank-diff-inv*:
$0 \leq \tau \Longrightarrow$ *diff-invariant* $(dI\ h_l\ h_h\ k)\ (f\ k)\ \{0..\tau\}\ UNIV\ 0\ Guard$
apply(*pred-simp*, *intro diff-invariant-conj-rule*)
apply(*force intro*!: *poly-derivatives diff-invariant-rules*)
apply(*rule-tac* $\nu'=\lambda t.\ 0$ **and** $\mu'=\lambda t.\ 1$ **in** *diff-invariant-leq-rule*, *simp-all*)
apply(*rule-tac* $\nu'=\lambda t.\ 0$ **and** $\mu'=\lambda t.\ 0$ **in** *diff-invariant-leq-rule*, *simp-all*)
by (*auto intro*!: *poly-derivatives diff-invariant-rules*)

lemma *tank-inv-arith1*:
assumes $0 \leq (\tau{::}real)$ **and** $c_o < c_i$ **and** b: $h_l \leq y_0$ **and** g: $\tau \leq (h_h - y_0) / (c_i - c_o)$
shows $h_l \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq h_h$
proof−

have $(c_i - c_o) \cdot \tau \leq (h_h - y_0)$
using *g assms*(*2*,*3*) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
thus $(c_i - c_o) \cdot \tau + y_0 \leq h_h$
by *auto*
show $h_l \leq (c_i - c_o) \cdot \tau + y_0$
using *b assms*(*1*,*2*) **by** (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)
qed

lemma *tank-inv-arith2*:
assumes $0 \leq (\tau{::}real)$ **and** $0 < c_o$ **and** b: $y_0 \leq h_h$ **and** g: $\tau \leq - ((h_l - y_0) / c_o)$
shows $h_l \leq y_0 - c_o \cdot \tau$ **and** $y_0 - c_o \cdot \tau \leq h_h$
proof−
have $\tau \cdot c_o \leq y_0 - h_l$
using *g* ‹$0 < c_o$› *pos-le-minus-divide-eq* **by** *fastforce*
thus $h_l \leq y_0 - c_o \cdot \tau$
by (*auto simp*: *mult.commute*)
show $y_0 - c_o \cdot \tau \leq h_h$
using *b assms*(*1*,*2*) **by** (*smt linordered-field-class.sign-simps*(*39*) *mult-less-cancel-right*)

qed

abbreviation *tank-dyn-dinv* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ (*real*ˆ*4*) *nd-fun* (*dyn*)
where $dyn\ c_i\ c_o\ h_l\ h_h\ \tau \equiv IF\ (\pi = 0)\ THEN$
$x\acute{}= f\ (c_i{-}c_o)\ \&\ G\ h_h\ (c_i{-}c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0\ DINV\ (dI\ h_l\ h_h\ (c_i{-}c_o))$
$ELSE\ x\acute{}= f\ ({-}c_o)\ \&\ G\ h_l\ ({-}c_o)\ on\ \{0..\tau\}\ UNIV\ @\ 0\ DINV\ (dI\ h_l\ h_h\ ({-}c_o))$

abbreviation $tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau \equiv LOOP\ (ctrl\ h_l\ h_h\ ;\ dyn\ c_i\ c_o\ h_l\ h_h\ \tau)\ INV\ (I\ h_l\ h_h)$

lemma *tank-inv*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\{I\ h_l\ h_h\}$ *tank-dinv* $c_i\ c_o\ h_l\ h_h\ \tau$ $\{I\ h_l\ h_h\}$
apply(*hyb-hoare* $\mathbf{U}(I\ h_l\ h_h \wedge t = 0 \wedge h_0 = h)$)
prefer *4* **prefer** *7* **using** *tank-diff-inv assms* **apply** *force*+
using *assms tank-inv-arith1 tank-inv-arith2* **by** *rel-auto′*

— Refined with invariants

lemma *R-tank-inv*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $[I\ h_l\ h_h,\ I\ h_l\ h_h] \geq tank\text{-}dinv\ c_i\ c_o\ h_l\ h_h\ \tau$
proof−
have $[I\ h_l\ h_h,\ I\ h_l\ h_h] \geq LOOP\ ((t ::= 0);[I\ h_l\ h_h \wedge t = 0,\ I\ h_l\ h_h])\ INV\ I\ h_l\ h_h$ (**is** - ≥ *?R*)
by (*refinement*, *rel-auto′*)
moreover have

?R $\geq$ *LOOP* ((*t* ::= *0*);(h_0 ::= *h*);[*I* h_l h_h $\wedge$ *t* = *0* $\wedge$ h_0 = *h*, *I* h_l h_h]) *INV I* h_l h_h (**is** - $\geq$ *?R*)
by (*refinement*, *rel-auto'*)
moreover have
?R $\geq$ *LOOP* (*ctrl* h_l h_h;[*I* h_l h_h $\wedge$ *t* = *0* $\wedge$ h_0 = *h*, *I* h_l h_h]) *INV I* h_l h_h (**is** - $\geq$ *?R*)
by (*simp only*: *mult.assoc*, *refinement*; (*force*)*?*, (*rule R-assign-law*)*?*) *rel-auto'*
moreover have
?R $\geq$ *LOOP* (*ctrl* h_l h_h; *dyn* c_i c_o h_l h_h τ) *INV I* h_l h_h
apply(*simp only*: *mult.assoc*, *refinement*; (*simp*)*?*)
prefer *4* **using** *tank-diff-inv assms* **apply** *force+*
using *tank-inv-arith1 tank-inv-arith2 assms* **by** *rel-auto'*
ultimately show [*I* h_l h_h, *I* h_l h_h] $\geq$ *tank-dinv* c_i c_o h_l h_h τ
by *auto*
qed

no-notation *ftank* (*f*)
and *tank-flow* (φ)
and *tank-guard* (*G*)
and *tank-loop-inv* (*I*)
and *tank-diff-inv* (*dI*)
and *tank-ctrl* (*ctrl*)
and *tank-dyn-dinv* (*dyn*)

end

Références

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1) :3–34, 1995.