

# Verification of Hybrid Systems

Jonathan Julián Huerta y Munive

October 8, 2020

## Abstract

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

# Contents

|  |          |
|--|----------|
| <b>Abstract</b>  | <b>2</b> |
| 0.1 Introductory Remarks   | 5        |
| 0.2 Hybrid Systems Preliminaries                                     | 5        |
| 0.2.1 Real numbers   | 6        |
| 0.2.2 Single variable derivatives                                    | 6        |
| 0.2.3 Intermediate Value Theorem                                     | 8        |
| 0.2.4 Filters  | 9        |
| 0.2.5 Multivariable derivatives                                      | 9        |
| 0.3 Ordinary Differential Equations                                  | 11       |
| 0.3.1 Initial value problems and orbits                              | 11       |
| 0.3.2 Differential Invariants  | 12       |
| 0.3.3 Picard-Lindelof  | 16       |
| 0.3.4 Flows for ODEs   | 19       |
| 0.4 Verification components for hybrid systems                       | 23       |
| 0.4.1 Verification of regular programs                               | 23       |
| 0.4.2 Verification of hybrid programs                                | 25       |
| 0.4.3 Derivation of the rules of dL                                  | 27       |
| 0.4.4 Examples   | 29       |
| 0.5 Verification components with predicate transformers              | 35       |
| 0.5.1 Verification of regular programs                               | 35       |
| 0.5.2 Verification of hybrid programs                                | 37       |
| 0.5.3 Derivation of the rules of dL                                  | 39       |
| 0.5.4 Examples   | 42       |
| 0.6 Verification components with MKA                                 | 46       |
| 0.6.1 Verification in AKA  | 46       |
| 0.6.2 Relational model   | 49       |
| 0.6.3 Store and weakest preconditions                                | 51       |
| 0.6.4 Verification of hybrid programs                                | 52       |
| 0.6.5 Derivation of the rules of dL                                  | 54       |
| 0.6.6 Examples   | 56       |
| 0.7 Verification components with MKA and non-deterministic functions | 61       |
| 0.7.1 Non-deterministic functions                                    | 61       |
| 0.7.2 Store and weakest preconditions                                | 62       |
| 0.7.3 Verification of hybrid programs                                | 64       |
| 0.7.4 Derivation of the rules of dL                                  | 65       |
| 0.7.5 Examples   | 68       |
| 0.8 Verification components with KAT                                 | 73       |
| 0.8.1 Hoare logic in KAT   | 73       |
| 0.8.2 refinement KAT   | 75       |
| 0.9 Verification and refinement of HS in the relational KAT          | 77       |
| 0.9.1 Relational model   | 78       |
| 0.9.2 Relational Store   | 79       |

|        |   |     |
|--------|---|-----|
| 0.9.3  | Verification of hybrid programs . . . . .                         | 81  |
| 0.9.4  | Refinement Components . . . . .                                   | 83  |
| 0.9.5  | Derivation of the rules of dL . . . . .                           | 86  |
| 0.9.6  | Examples . . . . .  | 87  |
| 0.10   | Verification and refinement of HS in the relational KAT . . . . . | 98  |
| 0.10.1 | Store and Hoare triples . . . . .                                 | 98  |
| 0.10.2 | Verification of hybrid programs . . . . .                         | 101 |
| 0.10.3 | Refinement Components . . . . .                                   | 103 |
| 0.10.4 | Derivation of the rules of dL . . . . .                           | 106 |
| 0.10.5 | Examples . . . . .  | 107 |
| 0.11   | Mathematical Preliminaries . . . . .                              | 117 |
| 0.11.1 | Syntax . . . . .  | 117 |
| 0.11.2 | Topology and sets . . . . .                                       | 117 |
| 0.11.3 | Functions . . . . .   | 118 |
| 0.11.4 | Suprema . . . . .   | 118 |
| 0.11.5 | Real numbers . . . . .  | 119 |
| 0.11.6 | Vectors and matrices . . . . .                                    | 120 |
| 0.11.7 | Diagonalization . . . . .   | 121 |
| 0.12   | Matrix norms . . . . .  | 123 |
| 0.12.1 | Matrix operator norm . . . . .                                    | 124 |
| 0.12.2 | Matrix maximum norm . . . . .                                     | 126 |
| 0.13   | Square Matrices . . . . .   | 128 |
| 0.13.1 | Definition . . . . .  | 128 |
| 0.13.2 | Ring of square matrices . . . . .                                 | 129 |
| 0.13.3 | Real normed vector space of square matrices . . . . .             | 131 |
| 0.13.4 | Real normed algebra of square matrices . . . . .                  | 133 |
| 0.13.5 | Banach space of square matrices . . . . .                         | 135 |
| 0.13.6 | Examples . . . . .  | 137 |
| 0.14   | Affine systems of ODEs . . . . .                                  | 140 |
| 0.14.1 | Existence and uniqueness for affine systems . . . . .             | 140 |
| 0.14.2 | Flow for affine systems . . . . .                                 | 142 |
| 0.15   | Verification examples . . . . .                                   | 145 |
| 0.15.1 | Examples . . . . .  | 145 |
| 0.16   | Examples . . . . .  | 148 |
| 0.16.1 | Basic . . . . .   | 149 |
| 0.16.2 | Advanced . . . . .  | 175 |
| 0.16.3 | ETCS: Proposition 1 (Controllability) . . . . .                   | 177 |
| 0.16.4 | ETCS: Proposition 4 (Reactivity) . . . . .                        | 177 |
| 0.17   | PID Examples . . . . .  | 178 |
| 0.17.1 | PI controller on kinematics . . . . .                             | 179 |
| 0.18   | Verification components with Kleene Algebras . . . . .            | 182 |
| 0.18.1 | Hoare logic and refinement in KAT . . . . .                       | 183 |
| 0.18.2 | refinement KAT . . . . .  | 185 |
| 0.18.3 | Verification in AKA (KAD) . . . . .                               | 187 |
| 0.18.4 | Relational model . . . . .  | 187 |
| 0.18.5 | State transformer model . . . . .                                 | 188 |
| 0.19   | VC_diffKAD . . . . .  | 190 |
| 0.19.1 | Stack Theories Preliminaries: VC_KAD and ODEs . . . . .           | 190 |
| 0.19.2 | VC_diffKAD Preliminaries . . . . .                                | 192 |
| 0.19.3 | Phase Space Relational Semantics . . . . .                        | 200 |
| 0.19.4 | Derivation of Differential Dynamic Logic Rules . . . . .          | 202 |
| 0.19.5 | Rules Testing . . . . .   | 215 |

## 0.1 Introductory Remarks

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

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

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

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

## 0.2 Hybrid Systems Preliminaries

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

**theory** *HS-Preliminaries*

**imports** *Ordinary-Differential-Equations.Picard-Lindelof-Qualitative*  
**begin**

— Syntax

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

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

### 0.2.1 Real numbers

**lemma** *abs-le-eq*:

**shows**  $(r::\text{real}) > 0 \implies (|x| < r) = (-r < x \wedge x < r)$   
**and**  $(r::\text{real}) > 0 \implies (|x| \leq r) = (-r \leq x \wedge x \leq r)$   
**by** *linarith+*

**lemma** *real-ivl-eqs*:

**assumes**  $0 < r$   
**shows**  $\text{ball } x \ r = \{x-r <--< x+r\}$  **and**  $\{x-r <--< x+r\} = \{x-r <..< x+r\}$   
**and**  $\text{ball } (r / 2) \ (r / 2) = \{0 <--< r\}$  **and**  $\{0 <--< r\} = \{0 <..< r\}$   
**and**  $\text{ball } 0 \ r = \{-r <--< r\}$  **and**  $\{-r <--< r\} = \{-r <..< r\}$   
**and**  $\text{cball } x \ r = \{x-r--x+r\}$  **and**  $\{x-r--x+r\} = \{x-r..x+r\}$   
**and**  $\text{cball } (r / 2) \ (r / 2) = \{0--r\}$  **and**  $\{0--r\} = \{0..r\}$   
**and**  $\text{cball } 0 \ r = \{-r--r\}$  **and**  $\{-r--r\} = \{-r..r\}$   
**unfolding** *open-segment-eq-real-ivl closed-segment-eq-real-ivl*  
**using** *assms* **by**  $(\text{auto simp: cball-def ball-def dist-norm field-simps})$

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

**lemma** *norm-rotate-eq*[*simp*]:

**fixes**  $x :: 'a::\{\text{banach}, \text{real-normed-field}\}$   
**shows**  $(x * \cos t - y * \sin t)^2 + (x * \sin t + y * \cos t)^2 = x^2 + y^2$   
**and**  $(x * \cos t + y * \sin t)^2 + (y * \cos t - x * \sin t)^2 = x^2 + y^2$

**proof**—

**have**  $(x * \cos t - y * \sin t)^2 = x^2 * (\cos t)^2 + y^2 * (\sin t)^2 - 2 * (x * \cos t) * (y * \sin t)$   
**by**  $(\text{simp add: power2-diff power-mult-distrib})$   
**also have**  $(x * \sin t + y * \cos t)^2 = y^2 * (\cos t)^2 + x^2 * (\sin t)^2 + 2 * (x * \cos t) * (y * \sin t)$   
**by**  $(\text{simp add: power2-sum power-mult-distrib})$   
**ultimately show**  $(x * \cos t - y * \sin t)^2 + (x * \sin t + y * \cos t)^2 = x^2 + y^2$   
**by**  $(\text{simp add: Groups.mult-ac(2) Groups.mult-ac(3) right-diff-distrib sin-squared-eq})$   
**thus**  $(x * \cos t + y * \sin t)^2 + (y * \cos t - x * \sin t)^2 = x^2 + y^2$   
**by**  $(\text{simp add: add.commute add.left-commute power2-diff power2-sum})$

**qed**

### 0.2.2 Single variable derivatives

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

**declare** *has-vderiv-on-const* [*poly-derivatives*]

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

**lemma** *vderiv-on-composeI*:

**assumes**  $D f = f' \text{ on } g \text{ ' } T$   
**and**  $D g = g' \text{ on } T$   
**and**  $h = (\lambda t. g' t *_{\mathbb{R}} f' (g t))$   
**shows**  $D (\lambda t. f (g t)) = h \text{ on } T$   
**apply**  $(\text{subst ssubst[of } h], \text{ simp})$   
**using** *assms* *has-vderiv-on-compose* **by** *auto*

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

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

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

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

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

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

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

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

— Examples for checking derivatives

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

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

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

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

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

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

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

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

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

by (auto intro!: poly-derivatives simp: power2-eq-square)

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

by (auto intro!: poly-derivatives simp: power2-eq-square)

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

by (auto intro!: poly-derivatives simp: power2-eq-square)

lemma  $c \neq 0 \implies D (\lambda t. \exp (a * \sin (\cos (t^4) / c))) =$   
 $(\lambda t. - 4 * a * t^3 * \sin (t^4) / c * \cos (\cos (t^4) / c) * \exp (a * \sin (\cos (t^4) / c)))$  on  $T$

by (intro poly-derivatives) (auto intro!: poly-derivatives simp: power2-eq-square)

### 0.2.3 Intermediate Value Theorem

lemma *IVT-two-functions*:

fixes  $f :: ('a::\{\text{linear-continuum-topology, real-vector}\}) \Rightarrow$

$('b::\{\text{linorder-topology, real-normed-vector, ordered-ab-group-add}\})$

assumes *conts*: continuous-on  $\{a..b\}$   $f$  continuous-on  $\{a..b\}$   $g$

and *ahyp*:  $f a < g a$  and *bhyp*:  $g b < f b$  and  $a \leq b$

shows  $\exists x \in \{a..b\}. f x = g x$

proof—

let  $?h x = f x - g x$

have  $?h a \leq 0$  and  $?h b \geq 0$

using *ahyp* *bhyp* by *simp-all*

also have continuous-on  $\{a..b\}$   $?h$

using *conts* continuous-on-diff by *blast*

ultimately obtain  $x$  where  $a \leq x \leq b$  and  $?h x = 0$

using *IVT*[of  $?h$ ]  $\langle a \leq b \rangle$  by *blast*

thus *?thesis*

using  $\langle a \leq b \rangle$  by *auto*

qed

lemma *IVT-two-functions-real-ivl*:

fixes  $f :: \text{real} \Rightarrow \text{real}$

assumes *conts*: continuous-on  $\{a--b\}$   $f$  continuous-on  $\{a--b\}$   $g$

and *ahyp*:  $f a < g a$  and *bhyp*:  $g b < f b$

shows  $\exists x \in \{a--b\}. f x = g x$

proof(cases  $a \leq b$ )

case *True*

then show *?thesis*

using *IVT-two-functions* *assms*

unfolding *closed-segment-eq-real-ivl* by *auto*

next

case *False*

hence  $a \geq b$

by *auto*

hence continuous-on  $\{b..a\}$   $f$  continuous-on  $\{b..a\}$   $g$

using *conts* *False* unfolding *closed-segment-eq-real-ivl* by *auto*

hence  $\exists x \in \{b..a\}. g x = f x$

using *IVT-two-functions*[of  $b a g f$ ] *assms*(3,4) *False* by *auto*

then show *?thesis*

using  $\langle a \geq b \rangle$  unfolding *closed-segment-eq-real-ivl* by *auto* force

qed



### 0.2.4 Filters

**lemma** *eventually-at-within-mono*:

**assumes**  $t \in \text{interior } T$  **and**  $T \subseteq S$   
**and** *eventually*  $P$  (at  $t$  within  $T$ )  
**shows** *eventually*  $P$  (at  $t$  within  $S$ )  
**by** (*meson assms eventually-within-interior interior-mono subsetD*)

**lemma** *netlimit-at-within-mono*:

**fixes**  $t::'a::\{\text{perfect-space}, \text{t2-space}\}$   
**assumes**  $t \in \text{interior } T$  **and**  $T \subseteq S$   
**shows**  $\text{netlimit}$  (at  $t$  within  $S$ ) =  $t$   
**using** *assms(1) interior-mono[OF  $T \subseteq S$ ] netlimit-within-interior* **by** *auto*

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

**assumes**  $(t::\text{real}) \in \text{interior } T$  **and**  $T \subseteq S$   
**and**  $D f \mapsto f'$  at  $t$  within  $T$   
**shows**  $D f \mapsto f'$  at  $t$  within  $S$   
**using** *assms(3) apply(unfold has-derivative-def tendsto-iff, safe)*  
**unfolding** *netlimit-at-within-mono[OF assms(1,2)] netlimit-within-interior[OF assms(1)]*  
**by** (*rule eventually-at-within-mono[OF assms(1,2)] simp*)

**lemma** *eventually-all-finite2*:

**fixes**  $P :: ('a::\text{finite}) \Rightarrow 'b \Rightarrow \text{bool}$   
**assumes**  $h:\forall i. \text{eventually } (P i) F$   
**shows** *eventually*  $(\lambda x. \forall i. P i x) F$   
**proof**(*unfold eventually-def*)  
**let**  $?F = \text{Rep-filter } F$   
**have**  $\text{obs}:\forall i. ?F (P i)$   
**using**  $h$  **by** *auto*  
**have**  $?F (\lambda x. \forall i \in \text{UNIV}. P i x)$   
**apply**(*rule finite-induct*)  
**by**(*auto intro: eventually-conj simp: obs h*)  
**thus**  $?F (\lambda x. \forall i. P i x)$   
**by** *simp*

**qed**

**lemma** *eventually-all-finite-mono*:

**fixes**  $P :: ('a::\text{finite}) \Rightarrow 'b \Rightarrow \text{bool}$   
**assumes**  $h1:\forall i. \text{eventually } (P i) F$   
**and**  $h2:\forall x. (\forall i. (P i x)) \longrightarrow Q x$   
**shows** *eventually*  $Q F$

**proof**—

**have** *eventually*  $(\lambda x. \forall i. P i x) F$   
**using**  $h1$  *eventually-all-finite2* **by** *blast*  
**thus** *eventually*  $Q F$   
**unfolding** *eventually-def*  
**using**  $h2$  *eventually-mono* **by** *auto*

**qed**

### 0.2.5 Multivariable derivatives

**lemma** *frechet-vec-lambda*:

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

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

**lemma** *tendsto-norm-bound*:

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

**lemma** *tendsto-zero-norm-bound*:

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

**lemma** *frechet-vec-nth*:

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

**lemma** *has-derivative-vec-lambda*:

**fixes**  $f::\text{real} \Rightarrow ('a::\text{banach})^{('n::\text{finite})}$   
**assumes**  $\forall i. D \ (\lambda t. f \ t \$ i) \mapsto (\lambda h. h *_R f' \ x \$ i) \text{ (at } x \text{ within } T)$

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

**lemma** *has-derivative-vec-nth*:

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

**lemma** *has-vderiv-on-vec-eq[simp]*:

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

**end**

## 0.3 Ordinary Differential Equations

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

**theory** *HS-ODEs*

**imports** *HS-Preliminaries*

**begin**

### 0.3.1 Initial value problems and orbits

**notation** *image* ( $\mathcal{P}$ )

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

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

**lemma** *ivp-solsI*:

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

**lemma** *ivp-solsD*:

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

**lemma** *in-ivp-sols-subset*:

$t_0 \in (U s) \implies (U s) \subseteq (T s) \implies X \in Sols f T S t_0 s \implies X \in Sols f U S t_0 s$

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

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

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

**lemma** *g-orbit-eq*:

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

**definition** *g-orbital* ::  $(\text{real} \Rightarrow \text{'a} \Rightarrow \text{'a}) \Rightarrow (\text{'a} \Rightarrow \text{bool}) \Rightarrow (\text{'a} \Rightarrow \text{real set}) \Rightarrow \text{'a set} \Rightarrow \text{real} \Rightarrow$   
 $(\text{'a}::\text{real-normed-vector}) \Rightarrow \text{'a set}$   
**where** *g-orbital f G U S t<sub>0</sub> s*  $= \bigcup \{\gamma X G (U s) \mid X. X \in \text{ivp-sols } f U S t_0 s\}$

**lemma** *g-orbital-eq*: *g-orbital f G U S t<sub>0</sub> s* =

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

**lemma** *g-orbitalI*:

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

**lemma** *g-orbitalD*:

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

**lemma** *g-orbital f G U S t<sub>0</sub> s* =  $\{X t \mid t X. X t \in \gamma X G (U s) \wedge X \in \text{Sols } f U S t_0 s\}$   
**unfolding** *g-orbital-eq g-orbit-eq* **by** *auto*

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

**lemma** *g-orbital f G U S t<sub>0</sub> s*  $\subseteq \text{g-orbital } f (\lambda s. \text{True}) U S t_0 s$   
**unfolding** *g-orbital-eq* **by** *auto*

**no-notation** *g-orbit* ( $\gamma$ )

### 0.3.2 Differential Invariants

**definition** *diff-invariant* ::  $(\text{'a} \Rightarrow \text{bool}) \Rightarrow (\text{real} \Rightarrow (\text{'a}::\text{real-normed-vector}) \Rightarrow \text{'a}) \Rightarrow$   
 $(\text{'a} \Rightarrow \text{real set}) \Rightarrow \text{'a set} \Rightarrow \text{real} \Rightarrow (\text{'a} \Rightarrow \text{bool}) \Rightarrow \text{bool}$   
**where** *diff-invariant I f U S t<sub>0</sub> G*  $\equiv (\bigcup \circ (\mathcal{P} (\text{g-orbital } f G U S t_0))) \{s. I s\} \subseteq \{s. I s\}$

**lemma** *diff-invariant-eq*: *diff-invariant I f U S t<sub>0</sub> G* =

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

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

*diff-invariant I f U S t<sub>0</sub> G* =  $(\forall s. I s \longrightarrow (\text{g-orbital } f G U S t_0 s) \subseteq \{s. I s\})$   
**unfolding** *diff-invariant-eq g-orbital-eq image-le-pred* **by** *auto*

**lemma** *diff-invariant I f U S t<sub>0</sub> (\lambda s. True)*  $\implies \text{diff-invariant } I f U S t_0 G$

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

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

**lemma** *diff-invariant-eq-rule* [*diff-invariant-rules*]:

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

**shows** *diff-invariant*  $(\lambda s. \mu\ s = \nu\ s)\ f\ U\ S\ t_0\ G$

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

**fix** *X t*

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

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

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

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

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

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

**then obtain**  $\tau$  **where**  $\mu\ (X\ t) - \nu\ (X\ t) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (t - t_0) *_R\ 0$

**using** *mvt-very-simple-closed-segmentE* **by** *blast*

**thus**  $\mu\ (X\ t) = \nu\ (X\ t)$

**by** (*simp add: xivp*(2))

**qed**

**lemma** *diff-invariant-leq-rule* [*diff-invariant-rules*]:

**fixes**  $\mu::'a::\text{banach} \Rightarrow \text{real}$

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

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

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

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

**shows** *diff-invariant*  $(\lambda s. \nu\ s \leq \mu\ s)\ f\ U\ S\ t_0\ G$

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

**fix** *X t* **assume** *Ghyp*:  $\forall \tau. \tau \in U\ (X\ t_0) \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$

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

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

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

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

**by** (*force,metis PiE*  $\langle X\ t_0 \in S \implies \{t_0 \dashv\dashv t\} \subseteq U\ (X\ t_0) \rangle$  *dual-order.trans*)

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

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

**{assume**  $t \neq t_0$

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

**and**  $(\mu(X\ t) - \nu(X\ t)) - (\mu(X\ t_0) - \nu(X\ t_0)) = (\lambda\tau. \tau * (\mu'(X\ r) - \nu'(X\ r)))\ (t - t_0)$

**using** *mvt-simple-closed-segmentE* *obs2* **by** *blast*

**hence** *mvt*:  $\mu(X\ t) - \nu(X\ t) = (t - t_0) * (\mu'(X\ r) - \nu'(X\ r)) + (\mu(X\ t_0) - \nu(X\ t_0))$

**by** *force*

**have** *primed*:  $\bigwedge \tau. \tau \in U\ (X\ t_0) \implies \tau > t_0 \implies G\ (X\ \tau) \implies \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$

$\bigwedge \tau. \tau \in U\ (X\ t_0) \implies \tau < t_0 \implies \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$

**using** *Gg*[*OF xivp*(1)] *Gl*[*OF xivp*(1)] **by** *auto*

**have**  $t > t_0 \implies r > t_0 \wedge G\ (X\ r) \wedge t_0 \leq t \implies r < t_0\ r \in U\ (X\ t_0)$

**using**  $\langle r \in \{t_0 < \dashv\dashv t\} \rangle$  *obs1* *Ghyp*

**unfolding** *open-segment-eq-real-ivl* *closed-segment-eq-real-ivl* **by** *auto*

**moreover** **have**  $r > t_0 \implies G\ (X\ r) \implies (\mu'(X\ r) - \nu'(X\ r)) \geq 0\ r < t_0 \implies (\mu'(X\ r) - \nu'(X\ r)) \leq 0$

**using** *primed*(1,2)[*OF*  $\langle r \in U\ (X\ t_0) \rangle$ ] **by** *auto*

**ultimately** **have**  $(t - t_0) * (\mu'(X\ r) - \nu'(X\ r)) \geq 0$

**by** (*case-tac*  $t \geq t_0$ , *force*, *auto simp: split-mult-pos-le*)

**hence**  $(t - t_0) * (\mu'(X\ r) - \nu'(X\ r)) + (\mu(X\ t_0) - \nu(X\ t_0)) \geq 0$

**using** *xivp*(2) **by** *auto*

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

lemma *diff-invariant-less-rule* [*diff-invariant-rules*]:

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

lemma *diff-invariant-nleq-rule*:

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

lemma *diff-invariant-neq-rule* [*diff-invariant-rules*]:

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

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

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

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

**lemma** *diff-invariant-conj-rule* [*diff-invariant-rules*]:

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

**lemma** *diff-invariant-disj-rule* [*diff-invariant-rules*]:

**assumes** *diff-invariant*  $I_1 f U S t_0 G$

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

### 0.3.3 Picard-Lindelof

A locale with the assumptions of Picard-Lindelof theorem. It extends *ll-on-open-it* by providing an initial time  $t_0 \in T$ .

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

**sublocale** *ll-on-open-it*  $T \ f \ S \ t_0$   
**by** (*unfold-locales*) (*auto simp: cont-vec-field lipschitz-vec-field interval-time open-domain*)

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

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

**abbreviation** *ex-ivl*  $s \equiv \text{existence-ivl } t_0 \ s$

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

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

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

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

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



**lemma unique-solution:** — proved for a subset of T for general applications

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

**proof—**

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

**qed**

Applications of lemma *unique-solution*:

**lemma unique-solution-closed-ivl:**

**assumes** *xivp*:  $D X = (\lambda t. f t (X t))$  **on**  $\{t_0 \dashv\dashv t\}$   $X t_0 = s$   $X \in \{t_0 \dashv\dashv t\} \rightarrow S$  **and**  $t \in T$   
**and** *yivp*:  $D Y = (\lambda t. f t (Y t))$  **on**  $\{t_0 \dashv\dashv t\}$   $Y t_0 = s$   $Y \in \{t_0 \dashv\dashv t\} \rightarrow S$  **and**  $s \in S$   
**shows**  $X t = Y t$   
**apply**(*rule unique-solution*[*OF*  $\langle s \in S \rangle$ , *of*  $\{t_0 \dashv\dashv t\}$ ], *simp-all add: assms*)  
**apply**(*unfold existence-ivl-def csols-eq ivp-sols-def, clarsimp*)  
**using** *xivp*  $\langle t \in T \rangle$  **by** *blast*

**lemma solution-eq-flow:**

**assumes** *xivp*:  $D X = (\lambda t. f t (X t))$  **on** *ex-ivl*  $s$   $X t_0 = s$   $X \in \text{ex-ivl } s \rightarrow S$   
**and**  $t \in \text{ex-ivl } s$  **and**  $s \in S$   
**shows**  $X t = \text{flow } t_0 s t$   
**apply**(*rule unique-solution*[*OF*  $\langle s \in S \rangle$  *init-time-ex-ivl*  $\langle t \in \text{ex-ivl } s \rangle$ ])  
**using** *flow-has-vderiv-on-ex-ivl flow-funcset-ex-ivl*  $\langle s \in S \rangle$  **by** (*auto simp: assms*)

**lemma ivp-unique-solution:**

**assumes**  $s \in S$  **and** *ivl*: *is-interval*  $(U s)$  **and**  $U s \subseteq T$  **and**  $t \in U s$   
**and** *ivp1*:  $Y_1 \in \text{Sols } f U S t_0 s$  **and** *ivp2*:  $Y_2 \in \text{Sols } f U S t_0 s$   
**shows**  $Y_1 t = Y_2 t$

**proof**(*rule unique-solution*[*OF*  $\langle s \in S \rangle$ , *of*  $\{t_0 \dashv\dashv t\}$ ], *simp-all*)

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

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

**lemma** *g-orbital-orbit*:

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

**proof**—

**have**  $\text{eq1}: \forall Z \in \text{Sols } f \ U \ S \ t_0 \ s. \forall t \in U \ s. Z \ t = Y \ t$   
**by** (*clarsimp*, *rule ivp-unique-solution*[*OF assms*(1,2,3) - - *ivp*], *auto*)  
**have**  $g\text{-orbital } f \ G \ U \ S \ t_0 \ s \subseteq g\text{-orbit } (\lambda t. Y \ t) \ G \ (U \ s)$

**proof**

**fix**  $x$  **assume**  $x \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$

**then obtain**  $Z$  **and**  $t$

**where** *z-def*:  $x = Z \ t \wedge t \in U \ s \wedge (\forall \tau \in \text{down } (U \ s) \ t. G \ (Z \ \tau)) \wedge Z \in \text{Sols } f \ U \ S \ t_0 \ s$

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

**hence**  $\{t_0 \dashv\dashv t\} \subseteq U \ s$

**using** *closed-segment-subset-interval*[*OF ivl ivp-solsD*(4)][*OF ivp*] **by** *blast*

**hence**  $\forall \tau \in \{t_0 \dashv\dashv t\}. Z \ \tau = Y \ \tau$

**using** *z-def* **apply** *clarsimp*

**by** (*rule ivp-unique-solution*[*OF assms*(1,2,3) - - *ivp*], *auto*)

**thus**  $x \in g\text{-orbit } Y \ G \ (U \ s)$

**using** *z-def eq1* **unfolding** *g-orbit-eq* **by** *simp metis*

**qed**

**moreover have**  $g\text{-orbit } Y \ G \ (U \ s) \subseteq g\text{-orbital } f \ G \ U \ S \ t_0 \ s$

**apply**(*unfold g-orbital-eq g-orbit-eq ivp-sols-def*, *clarsimp*)

**apply**(*rule-tac x=t in exI*, *rule-tac x=Y in exI*)

**using** *ivp-solsD*[*OF ivp*] **by** *auto*

**ultimately show** *?thesis*

**by** *blast*

**qed**

**end**

**lemma** *local-lipschitz-add*:

**fixes**  $f1 \ f2 :: \text{real} \Rightarrow 'a :: \text{banach} \Rightarrow 'a$

**assumes** *local-lipschitz*  $T \ S \ f1$

**and** *local-lipschitz*  $T \ S \ f2$

**shows** *local-lipschitz*  $T \ S \ (\lambda t \ s. f1 \ t \ s + f2 \ t \ s)$

**proof**(*unfold local-lipschitz-def*, *clarsimp*)

**fix**  $s$  **and**  $t$  **assume**  $s \in S$  **and**  $t \in T$

**obtain**  $\varepsilon_1 \ L1$  **where**  $\varepsilon_1 > 0$  **and**  $L1: \bigwedge \tau. \tau \in \text{cball } t \ \varepsilon_1 \cap T \implies L1\text{-lipschitz-on } (\text{cball } s \ \varepsilon_1 \cap S) \ (f1 \ \tau)$

**using** *local-lipschitzE*[*OF assms*(1)  $\langle t \in T \rangle \langle s \in S \rangle$ ] **by** *blast*

**obtain**  $\varepsilon_2 \ L2$  **where**  $\varepsilon_2 > 0$  **and**  $L2: \bigwedge \tau. \tau \in \text{cball } t \ \varepsilon_2 \cap T \implies L2\text{-lipschitz-on } (\text{cball } s \ \varepsilon_2 \cap S) \ (f2 \ \tau)$

**using** *local-lipschitzE*[*OF assms*(2)  $\langle t \in T \rangle \langle s \in S \rangle$ ] **by** *blast*

**have** *ballH*:  $\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S \subseteq \text{cball } s \ \varepsilon_1 \cap S \cap \text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S \subseteq \text{cball } s \ \varepsilon_2 \cap S$

**by** *auto*

**have** *obs1*:  $\forall \tau \in \text{cball } t \ \varepsilon_1 \cap T. L1\text{-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S) \ (f1 \ \tau)$

**using** *lipschitz-on-subset*[*OF L1 ballH*(1)] **by** *blast*

**also have** *obs2*:  $\forall \tau \in \text{cball } t \ \varepsilon_2 \cap T. L2\text{-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S) \ (f2 \ \tau)$

**using** *lipschitz-on-subset*[*OF L2 ballH*(2)] **by** *blast*

**ultimately have**  $\forall \tau \in \text{cball } t \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap T.$

$(L1 + L2)\text{-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S) \ (\lambda s. f1 \ \tau \ s + f2 \ \tau \ s)$

**using** *lipschitz-on-add* **by** *fastforce*

**thus**  $\exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap T. L\text{-lipschitz-on } (\text{cball } s \ u \cap S) \ (\lambda s. f1 \ t \ s + f2 \ t \ s)$

**apply**(*rule-tac x=min*  $\varepsilon_1 \ \varepsilon_2$  **in** *exI*)

**using**  $\langle \varepsilon_1 > 0 \rangle \langle \varepsilon_2 > 0 \rangle$  **by** *force*

**qed**

**lemma** *picard-lindelof-add*: *picard-lindelof*  $f1\ T\ S\ t_0 \implies \text{picard-lindelof } f2\ T\ S\ t_0 \implies$   
*picard-lindelof*  $(\lambda t\ s. f1\ t\ s + f2\ t\ s)\ T\ S\ t_0$   
**unfolding** *picard-lindelof-def* **apply**(*clarsimp*, *rule conjI*)  
**using** *continuous-on-add* **apply** *fastforce*  
**using** *local-lipschitz-add* **by** *blast*

**lemma** *picard-lindelof-constant*: *picard-lindelof*  $(\lambda t\ s. c)\ UNIV\ UNIV\ t_0$   
**apply**(*unfold-locales*, *simp-all* *add*: *local-lipschitz-def* *lipschitz-on-def*, *clarsimp*)  
**by** (*rule-tac*  $x=1$  **in** *exI*, *clarsimp*, *rule-tac*  $x=1/2$  **in** *exI*, *simp*)

### 0.3.4 Flows for ODEs

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

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

**lemma** *in-ivp-sols-ivl*:  
**assumes**  $t \in T\ s \in S$   
**shows**  $(\lambda t. \varphi\ t\ s) \in \text{Sols}\ (\lambda t. f)\ (\lambda s. \{0--t\})\ S\ 0\ s$   
**apply**(*rule ivp-solsI*)  
**using** *ivp* *assms* **by** *auto*

**lemma** *eq-solution-ivl*:  
**assumes** *xivp*:  $D\ X = (\lambda t. f\ (X\ t))$  *on*  $\{0--t\}\ X\ 0 = s\ X \in \{0--t\} \rightarrow S$   
**and** *indom*:  $t \in T\ s \in S$   
**shows**  $X\ t = \varphi\ t\ s$   
**apply**(*rule unique-solution-closed-ivl*[*OF* *xivp*  $\langle t \in T \rangle$ ])  
**using**  $\langle s \in S \rangle$  *ivp* *indom* **by** *auto*

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

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

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

lemma *has-derivative-on-open2*:

assumes  $t < 0 \ t \in T \ s \in S$   
 obtains  $B$  where  $t \in B$  and *open*  $B$  and  $B \subseteq T$   
 and  $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ t \ s))$  at  $t$  within  $B$

proof—

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

lemma *has-derivative-on-open3*:

assumes  $s \in S$   
 obtains  $B$  where  $0 \in B$  and *open*  $B$  and  $B \subseteq T$   
 and  $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ 0 \ s))$  at  $0$  within  $B$

proof—

obtain  $r::\text{real}$  where *rHyp*:  $r > 0 \ \text{ball } 0 \ r \subseteq T$   
 using *open-contains-ball-eq* *open-domain(1)* *init-time* **by** *blast*  
 hence  $r/2 \in T \ -r/2 \in T \ r/2 > 0$   
 unfolding *real-ivl-eqs* **by** *auto*  
 hence *subs*:  $\{0 \dashv\dashv -r/2\} \subseteq T \ \{0 \dashv\dashv (-r/2)\} \subseteq T$

using *subintervalI*[*OF init-time*] by *auto*  
 hence  $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{0--r/2\}$   
 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{0--(-r/2)\}$   
 using *ivp(1)*[*OF -  $\langle s \in S \rangle$* ] by *auto*  
 also have  $\{0--r/2\} = \{0--r/2\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\}$   
 $\{0--(-r/2)\} = \{0--(-r/2)\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\}$   
 unfolding *closed-segment-eq-real-ivl*  $\langle r/2 > 0 \rangle$  by *auto*  
 ultimately have *vderivs*:  
 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{0--r/2\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\}$   
 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{0--(-r/2)\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\}$   
 unfolding *closed-segment-eq-real-ivl*  $\langle r/2 > 0 \rangle$  by *auto*  
 have *obs*:  $0 \in \{-r/2 <--< r/2\}$   
 unfolding *open-segment-eq-real-ivl* using  $\langle r/2 > 0 \rangle$  by *auto*  
 have *union*:  $\{-r/2--r/2\} = \{0--r/2\} \cup \{0--(-r/2)\}$   
 unfolding *closed-segment-eq-real-ivl* by *auto*  
 hence  $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{-r/2--r/2\}$   
 using *has-vderiv-on-union*[*OF vderivs*] by *simp*  
 hence  $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$  on  $\{-r/2 <--< r/2\}$   
 using *has-vderiv-on-subset*[*OF - segment-open-subset-closed*[*of -r/2 r/2*]] by *auto*  
 hence  $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ 0 \ s))$  (at 0 within  $\{-r/2 <--< r/2\}$ )  
 unfolding *has-vderiv-on-def* *has-vector-derivative-def* using *obs* by *blast*  
 moreover have *open*  $\{-r/2 <--< r/2\}$   
 unfolding *open-segment-eq-real-ivl* by *simp*  
 moreover have  $\{-r/2 <--< r/2\} \subseteq T$   
 using *subs union segment-open-subset-closed* by *blast*  
 ultimately show *?thesis*  
 using *obs that* by *blast*  
 qed

lemma *has-derivative-on-open*:

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

lemma *in-domain*:

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

lemma *has-vderiv-on-domain*:

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

lemma *in-ivp-sols*:

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

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

**lemma** *eq-solution*:

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

**lemma** *ivp-sols-collapse*:

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

**lemma** *additive-in-ivp-sols*:

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

**lemma** *is-monoid-action*:

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

**proof**—

**show**  $\varphi 0 s = s$   
**using** *ivp assms* **by** *simp*  
**have**  $\varphi (0 + t_2) s = \varphi t_2 s$   
**by** *simp*  
**also have**  $\varphi (0 + t_2) s \in S$   
**using** *in-domain assms* **by** *auto*  
**ultimately show**  $\varphi (t_1 + t_2) s = \varphi t_1 (\varphi t_2 s)$   
**using** *eq-solution*[ $OF$  - - - *additive-in-ivp-sols*] *assms* **by** *auto*

**qed**

**lemma** *g-orbital-collapses*:

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

**definition** *orbit* ::  $'a \Rightarrow 'a \text{ set } (\gamma^\varphi)$

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

**lemma** *orbit-eq*:

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

**lemma** *true-g-orbit-eq*:

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

**end**

**lemma** *line-is-local-flow*:

```

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

```

**end**

## 0.4 Verification components for hybrid systems

A light-weight version of the verification components. We define the forward box operator to compute weakest liberal preconditions (wlps) of hybrid programs. Then we introduce three methods for verifying correctness specifications of the continuous dynamics of a HS.

**theory** *HS-VC-Spartan*

**imports** *HS-ODEs*

**begin**

**type-synonym** 'a pred = 'a ⇒ bool

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

**notation** *Union* (μ)

**and** *g-orbital* ((1x'=- & - on - - @ -))

### 0.4.1 Verification of regular programs

First we add lemmas for computation of weakest liberal preconditions (wlps).

**definition** *fbox* :: ('a ⇒ 'b set) ⇒ 'b pred ⇒ 'a pred ([-] - [61,81] 82)

**where** |F| P = (λs. (∀ s'. s' ∈ F s ⟶ P s'))

**lemma** *fbox-iso*: P ≤ Q ⟹ |F| P ≤ |F| Q

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

**lemma** *fbox-anti*: ∀ s. F<sub>1</sub> s ⊆ F<sub>2</sub> s ⟹ |F<sub>2</sub>| P ≤ |F<sub>1</sub>| P

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

**lemma** *fbox-invariants*:

**assumes** I ≤ |F| I **and** J ≤ |F| J

**shows** (λs. I s ∧ J s) ≤ |F| (λs. I s ∧ J s)

**and** (λs. I s ∨ J s) ≤ |F| (λs. I s ∨ J s)

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

Now, we compute wlps for specific programs starting with *skip*.

**abbreviation** *skip* ≡ (λs. {s})

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

**unfolding** *fbox-def* **by** *simp*

**definition** *test* :: 'a pred ⇒ 'a ⇒ 'a set ((1<sub>?</sub>-?))

**where** <sub>?</sub>P? = (λs. {x. x = s ∧ P x})

**lemma** *fbox-test*: (|<sub>?</sub>P?| Q) s = (P s ⟶ Q s)

**unfolding** *fbox-def test-def* **by** *simp*

Next, we introduce assignments and their wlps.

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

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

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

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

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

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

The wlp of a (kleisli) composition is just the composition of the wlp.

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

**lemma**  $kcomp\text{-}eq: (f ; g)\ x = \bigcup \{g\ y \mid y. y \in f\ x\}$   
**unfolding**  $kcomp\text{-}def\ image\text{-}def$  **by**  $auto$

**lemma**  $fbox\text{-}kcomp[simp]: |G ; F| P = |G| |F| P$   
**unfolding**  $fbox\text{-}def\ kcomp\text{-}def$  **by**  $auto$

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

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

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

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

**lemma**  $hoare\text{-}if\text{-}then\text{-}else:$   
**assumes**  $(\lambda s. P\ s \wedge T\ s) \leq |X| Q$   
**and**  $(\lambda s. P\ s \wedge \neg T\ s) \leq |Y| Q$   
**shows**  $P \leq |IF\ T\ THEN\ X\ ELSE\ Y| Q$   
**using**  $assms$  **unfolding**  $fbox\text{-}def\ ifthenelse\text{-}def$  **by**  $auto$

The final wlp we add is that of the finite iteration.

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

**lemma**  $kpower\text{-}base:$   
**shows**  $kpower\ f\ 0\ s = \{s\}$  **and**  $kpower\ f\ (Suc\ 0)\ s = f\ s$   
**unfolding**  $kpower\text{-}def$  **by** ( $auto\ simp: kcomp\text{-}eq$ )

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



**unfolding** *kcomp-eq*  
**apply**(*induct n*)  
**unfolding** *kpower-base*  
**apply**(*force simp: subset-antisym*)  
**unfolding** *kpower-def kcomp-eq* **by** *simp*

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

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

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

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

**proof**—

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

**qed**

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

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

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

## 0.4.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

**context** *local-flow*  
**begin**

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

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

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

**lemma** *fbox-orbit*:  $|\gamma^\varphi] \ Q = (\lambda s. s \in S \longrightarrow (\forall t \in T. Q \ (\varphi \ t \ s)))$   
**unfolding** *orbit-def fbox-g-ode* **by** *simp*

**end**

Verification with differential invariants

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

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

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

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

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

**using** *assms* **unfolding** *fbox-diff-inv diff-invariant-eq image-le-pred* **by** *auto*

**context** *local-flow*

**begin**

**lemma** *fbox-diff-inv-eq*:

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

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

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

**end**

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

**unfolding** *g-ode-inv-def*  
**apply**(*rule-tac b=|x' = f \ \& \ G \text{ on } U S @ t\_0| I in order.trans*)  
**apply**(*rule-tac I=I in fbox-g-orbital-inv, simp-all*)  
**apply**(*subst fbox-g-orbital-guard, simp*)  
**by** (*rule fbox-iso, force*)

### 0.4.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

**abbreviation** *g-dl-orbit* ::  $((a::\text{banach}) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \Rightarrow 'a \text{ set}$

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

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

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

**lemma** *diff-solve-axiom1*:

**assumes** *local-flow f UNIV UNIV  $\varphi$*   
**shows**  $|x' = f \ \& \ G| Q =$   
 $(\lambda s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))$   
**by** (*subst local-flow.fbox-g-ode-subset[OF assms], auto*)

**lemma** *diff-solve-axiom2*:

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

**lemma** *diff-solve-rule*:

**assumes** *local-flow f UNIV UNIV  $\varphi$*   
**and**  $\forall s. P s \longrightarrow (\forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))$   
**shows**  $P \leq |x' = f \ \& \ G| Q$   
**using** *assms* **by**(*subst local-flow.fbox-g-ode-subset[OF assms(1)] auto*)

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

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

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

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

**proof**

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

**qed**

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

**fix**  $s$   
**{fix**  $s'$  **assume**  $s' \in (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s$   
**then obtain**  $\tau::\text{real}$  **and**  $X$  **where**  $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$   
**and**  $X \ \tau = s'$  **and**  $\tau \in U \ s$  **and**  $\text{guard-}x:\mathcal{P} \ X \ (\text{down } (U \ s) \ \tau) \subseteq \{s. \ G \ s\}$   
**using** *g-orbitalD[of s' f G U S t\_0 s]* **by** *blast*  
**have**  $\forall t \in (\text{down } (U \ s) \ \tau). \ \mathcal{P} \ X \ (\text{down } (U \ s) \ t) \subseteq \{s. \ G \ s\}$   
**using** *guard-x* **by** (*force simp: image-def*)  
**also have**  $\forall t \in (\text{down } (U \ s) \ \tau). \ t \in U \ s$   
**using**  $\langle \tau \in U \ s \rangle$  *closed-segment-subset-interval* **by** *auto*  
**ultimately have**  $\forall t \in (\text{down } (U \ s) \ \tau). \ X \ t \in (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s$   
**using** *g-orbitalI[OF x-ivp]* **by** (*metis (mono-tags, lifting)*)  
**hence**  $\forall t \in (\text{down } (U \ s) \ \tau). \ C \ (X \ t)$   
**using** *assms* **unfolding** *fbox-def* **by** *meson*  
**hence**  $s' \in (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ s$   
**using** *g-orbitalI[OF x-ivp \langle \tau \in U \ s \rangle guard-x \langle X \ \tau = s' \rangle]* **by** *fastforce*  
**thus**  $(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ s$   
**by** *blast*  
**next show**  $\bigwedge s. \ (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s$   
**by** (*auto simp: g-orbital-eq*)  
**qed**

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

hence  $X t \in (x' = f \ \& \ (\lambda s. G s \wedge C s))$  on  $U S @ t_0$   $s$   
 using *guard-x*  $\langle t \in U s \rangle$  **by** (*auto intro!*: *g-orbitalI x-ivp*)  
 thus  $Q (X t)$   
 using  $\langle P s \rangle$  *fbox-Q* **by** (*subst (asm) fbox-def*) *auto*  
**qed**

**lemma** *diff-inv-axiom1*:

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

**lemma** *diff-inv-axiom2*:

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

**proof**(*unfold fbox-g-orbital, subst fbox-def, clarsimp simp: fun-eq-iff*)

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

**qed**

**lemma** *diff-inv-rule*:

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

**end**

#### 0.4.4 Examples

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

**theory** *HS-VC-Examples*

**imports** *HS-VC-Spartan*

**begin**

## Pendulum

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

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

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

— Verified with annotated dynamics.

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

— Verified with differential invariants.

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

— Verified with the flow.

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

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

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

## Bouncing Ball

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

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

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

— Verified with differential invariants.

**named-theorems**  $bb\text{-real-arith}$  *real arithmetic properties for the bouncing ball.*

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

**have**  $v * v = 2 * g * x - 2 * g * h \wedge 0 > g$   
**using** *inv* **and**  $\langle 0 > g \rangle$  **by** *auto*  
**hence**  $obs: v * v = 2 * g * (x - h) \wedge 0 > g \wedge v * v \geq 0$   
**using** *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)  
**hence**  $(v * v) / (2 * g) = (x - h)$   
**by** *auto*  
**also from** *obs* **have**  $(v * v) / (2 * g) \leq 0$   
**using** *divide-nonneg-neg* **by** *fastforce*  
**ultimately have**  $h - x \geq 0$   
**by** *linarith*  
**thus** *?thesis* **by** *auto*  
**qed**

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

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

— Verified with annotated dynamics.

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

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

by (simp add: semiring-normalization-rules(29))  
 finally have ?rhs = ?middle.  
 ultimately show ?thesis by auto  
 qed

**lemma** *bouncing-ball-dyn*:  $g < 0 \implies h \geq 0 \implies$   
 $(\lambda s. s\$1 = h \wedge s\$2 = 0) \leq$   
 $|LOOP ($   
 $(EVOL (\varphi g) (\lambda s. s\$1 \geq 0) T) ;$   
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$   
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$   
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$   
 by (rule fbox-loopI) (auto simp: bb-real-arith)

— Verified with the flow.

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

**lemma** *bouncing-ball-flow*:  $g < 0 \implies h \geq 0 \implies$   
 $(\lambda s. s\$1 = h \wedge s\$2 = 0) \leq$   
 $|LOOP ($   
 $(x'=(\lambda t. f g) \ \& \ (\lambda s. s\$1 \geq 0) \text{ on } (\lambda s. UNIV) UNIV @ 0) ;$   
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$   
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)]$   
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$   
 apply(rule fbox-loopI, simp-all add: local-flow.fbox-g-ode-subset[OF local-flow-ball])  
 by (auto simp: bb-real-arith)

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

## Thermostat

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

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

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

— Verified with the flow.

**lemma** *norm-diff-temp-dyn*:  $0 < a \implies \|f \ a \ L \ s_1 - f \ a \ L \ s_2\| = |a| * |s_1\$1 - s_2\$1|$   
**proof**(simp add: norm-vec-def L2-set-def, unfold UNIV-4, simp)  
 assume a1:  $0 < a$   
 have f2:  $\bigwedge r \ ra. |(r::real) + -ra| = |ra + -r|$   
 by (metis abs-minus-commute minus-real-def)



**have**  $\bigwedge r \text{ ra } rb. (r::\text{real}) * ra + - (r * rb) = r * (ra + - rb)$   
**by** (*metis minus-real-def right-diff-distrib*)  
**hence**  $|a * (s_1\$1 + - L) + - (a * (s_2\$1 + - L))| = a * |s_1\$1 + - s_2\$1|$   
**using** *a1* **by** (*simp add: abs-mult*)  
**thus**  $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$   
**using** *f2 minus-real-def* **by** *presburger*  
**qed**

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

**lemma** *local-flow-temp*:  $a > 0 \implies \text{local-flow } (f \ a \ L) \text{ UNIV UNIV } (\varphi \ a \ L)$   
**by** (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-temp-dyn simp: forall-4 vec-eq-iff*)

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

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

```

thus  $(L - Tmax) \leq \exp(-(a * t)) * (L - T)$ 
by auto
thus  $L - \exp(-(a * t)) * (L - T) \leq Tmax$ 
by auto
show  $Tmin \leq L - \exp(-(a * t)) * (L - T)$ 
using Thyps and obs by auto
qed

```

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

**lemma** *thermostat*:

```

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

```

**no-notation** *temp-vec-field* (*f*)  
**and** *temp-flow* ( $\varphi$ )

**Tank**

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

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

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

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

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

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

**lemma** *tank-arith*:

```

assumes  $0 \leq (\tau::real)$  and  $0 < c_o$  and  $c_o < c_i$ 
shows  $\forall \tau \in \{0.. \tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$ 
and  $\forall \tau \in \{0.. \tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$ 

```

```

and  $hmin \leq y \implies hmin \leq (c_i - c_o) * \tau + y$ 
and  $y \leq hmax \implies y - c_o * \tau \leq hmax$ 
apply(simp-all add: field-simps le-divide-eq assms)
using assms apply (meson add-mono less-eq-real-def mult-left-mono)
using assms by (meson add-increasing2 less-eq-real-def mult-nonneg-nonneg)

```

**lemma** *tank-flow*:

```

assumes  $0 \leq \tau$  and  $0 < c_o$  and  $c_o < c_i$ 
shows  $I \ hmin \ hmax \leq$ 
|LOOP
  — control
  (( $2 ::= (\lambda s. 0)$ );( $3 ::= (\lambda s. s\$1)$ ));
  (IF ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ ) THEN ( $4 ::= (\lambda s. 1)$ ) ELSE
  (IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) THEN ( $4 ::= (\lambda s. 0)$ ) ELSE skip));
  — dynamics
  (IF ( $\lambda s. s\$4 = 0$ ) THEN ( $x' = (\lambda t. f \ (c_i - c_o)) \ \& \ G \ hmax \ (c_i - c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0$ )
  ELSE ( $x' = (\lambda t. f \ (-c_o)) \ \& \ G \ hmin \ (-c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0$ ) ) INV  $I \ hmin \ hmax$ ]
I hmin hmax
apply(rule fbox-loopI, simp-all add: le-fun-def)
apply(clarsimp simp: le-fun-def local-flow.fbox-g-ode-ivl[OF local-flow-tank assms(1) UNIV-I])
using assms tank-arith[OF - assms(2,3)] by auto

```

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

```

and tank-flow ( $\varphi$ )
and tank-loop-inv (I)
and tank-diff-inv (dI)
and tank-guard (G)

```

**end**

## 0.5 Verification components with predicate transformers

We use the categorical forward box operator  $fb_{\mathcal{F}}$  to compute weakest liberal preconditions (wlps) of hybrid programs. Then we repeat the three methods for verifying correctness specifications of the continuous dynamics of a HS.

**theory** *HS-VC-PT*

**imports** *../HS-ODEs Transformer-Semantics.Kleisli-Quantale*

**begin**

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

```

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

```

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

**notation** *eta* (*skip*)

```

and kcomp (infixl ; 75)
and g-orbital (( $1x' = - \ \& \ - \ on \ - \ @ \ -$ ))

```

### 0.5.1 Verification of regular programs

Properties of the forward box operator.

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

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

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

**lemma**  $ffb-iso: P \leq Q \implies fb_{\mathcal{F}} F P \leq fb_{\mathcal{F}} F Q$   
**unfolding**  $ffb-eq$  **by**  $auto$

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

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

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

Next, we introduce assignments and their wlp.

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

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

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

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

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

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

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

**lemma**  $ffb-kcomp[simp]: fb_{\mathcal{F}} (G ; F) P = fb_{\mathcal{F}} G (fb_{\mathcal{F}} F P)$   
**unfolding**  $ffb-eq$  **by**  $(auto simp: kcomp-def)$

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

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

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

**lemma**  $ffb-if-then-else[simp]$ :  
 $fb_{\mathcal{F}} (IF T THEN X ELSE Y) Q = \{s. T s \longrightarrow s \in fb_{\mathcal{F}} X Q\} \cap \{s. \neg T s \longrightarrow s \in fb_{\mathcal{F}} Y Q\}$

**unfolding** *ffb-eq ifthenelse-def* **by** *auto*

**lemma** *hoare-if-then-else*:

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

We also deal with finite iteration.

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

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

**lemma** *ffb-kstarI*:

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

**proof** –

**have**  $I \subseteq \text{fb}_{\mathcal{F}} (\text{kstar}\ F)\ I$   
**using** *assms(3) kstar-inv* **by** *blast*  
**hence**  $P \leq \text{fb}_{\mathcal{F}} (\text{kstar}\ F)\ I$   
**using** *assms(1)* **by** *auto*  
**also have**  $\text{fb}_{\mathcal{F}} (\text{kstar}\ F)\ I \leq \text{fb}_{\mathcal{F}} (\text{kstar}\ F)\ Q$   
**by** (*rule ffb-iso[OF assms(2)]*)  
**finally show** *?thesis* .

**qed**

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

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

**lemma** *ffb-loopI-break*:

$P \leq \text{fb}_{\mathcal{F}} Y\ \{s. I\ s\} \Longrightarrow \{s. I\ s\} \leq \text{fb}_{\mathcal{F}} X\ \{s. I\ s\} \Longrightarrow \{s. I\ s\} \leq Q \Longrightarrow P \leq \text{fb}_{\mathcal{F}} (Y ; (\text{LOOP}\ X\ \text{INV}\ I))\ Q$   
**by** (*rule hoare-kcomp, force*) (*rule ffb-loopI, auto*)

## 0.5.2 Verification of hybrid programs

Verification by providing evolution

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

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

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

Verification by providing solutions

**lemma** *ffb-g-orbital*:  $\text{fb}_{\mathcal{F}} (x' = f \ \&\ G\ \text{on}\ U\ S\ @\ t_0)\ Q =$

$\{s. \forall X \in \text{Sols } f \ U \ S \ t_0 \ s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G \ (X \ \tau)) \longrightarrow (X \ t) \in Q\}$   
**unfolding** *ffb-eq g-orbital-eq* **by** (*auto simp: fun-eq-iff*)

**context** *local-flow*

**begin**

**lemma** *ffb-g-ode-subset*:

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

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

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

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

**end**

Verification with differential invariants

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

**lemma** *ffb-g-orbital-guard*:

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

**lemma** *ffb-g-orbital-inv*:

**assumes**  $P \leq I$  **and**  $I \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ I$  **and**  $I \leq Q$   
**shows**  $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ Q$   
**using** *assms(1)*  
**apply**(*rule order.trans*)  
**using** *assms(2)*  
**apply**(*rule order.trans*)  
**by** (*rule ffb-iso[OF assms(3)]*)

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

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

**lemma** *bdf-diff-inv*:

$\text{diff-invariant } I \ f \ U \ S \ t_0 \ G = (\text{bdf}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ \{s. I \ s\} \leq \{s. I \ s\})$   
**unfolding** *ffb-fbd-galois-var* **by** (*auto simp: diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

**lemma** *diff-inv-guard-ignore*:

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

**context** *local-flow*

**begin**

**lemma** *ffb-diff-inv-eq*:

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

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

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

**end**

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

**unfolding** *g-ode-inv-def*  
**apply**(*rule-tac b=fb\_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U S @ t\_0) \{s. I s\} in order.trans*)  
**apply**(*rule-tac I={s. I s} in ffb-g-orbital-inv, simp-all*)  
**apply**(*subst ffb-g-orbital-guard, simp*)  
**by** (*rule ffb-iso, force*)

### 0.5.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

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

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

**lemma** *diff-solve-axiom1*:

**assumes** *local-flow f UNIV UNIV \varphi*  
**shows**  $fb_{\mathcal{F}} (x' = f \ \& \ G) Q =$   
 $\{s. \forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow (\varphi \ t \ s) \in Q\}$   
**by** (*subst local-flow.ffib-g-ode-subset[OF assms], auto*)

**lemma** *diff-solve-axiom2*:

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

**lemma** *diff-solve-rule*:

**assumes** *local-flow f UNIV UNIV \varphi*

and  $\forall s. s \in P \longrightarrow (\forall t \geq 0. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow (\varphi t s) \in Q)$   
 shows  $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q$   
 using *assms* **by** (*subst local-flow.ffb-g-ode-subset[OF assms(1)]*) *auto*

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

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

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

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

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

**qed**

**lemma** *diff-cut-axiom*:  
 assumes  $fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ \{s. \ C \ s\} = UNIV$   
 shows  $fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ Q = fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ Q$   
**proof** (*rule-tac f = \lambda x. fb\_{\mathcal{F}} x Q in HOL.arg-cong, rule ext, rule subset-antisym*)

fix  $s$   
 {fix  $s'$  **assume**  $s' \in (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s$   
 then **obtain**  $\tau :: \text{real}$  **and**  $X$  **where**  $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$   
 and  $X \ \tau = s'$  **and**  $\tau \in (U \ s)$  **and**  $\text{guard-}x:\mathcal{P} \ X \ (\text{down } (U \ s) \ \tau) \subseteq \{s. \ G \ s\}$   
 using *g-orbitalD[of s' f G - S t\_0 s]* **by** *blast*  
 have  $\forall t \in (\text{down } (U \ s) \ \tau). \ \mathcal{P} \ X \ (\text{down } (U \ s) \ t) \subseteq \{s. \ G \ s\}$   
 using *guard-x* **by** (*force simp: image-def*)  
 also **have**  $\forall t \in (\text{down } (U \ s) \ \tau). \ t \in (U \ s)$   
 using  $\langle \tau \in (U \ s) \rangle$  *closed-segment-subset-interval* **by** *auto*  
 ultimately **have**  $\forall t \in (\text{down } (U \ s) \ \tau). \ X \ t \in (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ s$   
 using *g-orbitalI[OF x-ivp]* **by** (*metis (mono-tags, lifting)*)  
 hence  $\forall t \in (\text{down } (U \ s) \ \tau). \ C \ (X \ t)$   
 using *assms* **unfolding** *ffb-eq* **by** *fastforce*  
 hence  $s' \in (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ s$   
 using *g-orbitalI[OF x-ivp]*  $\langle \tau \in (U \ s) \rangle$  *guard-x*  $\langle X \ \tau = s' \rangle$   
**unfolding** *image-le-pred* **by** *fastforce*}

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

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

**qed**

**lemma** *diff-cut-rule*:  
 assumes *ffb-C*:  $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ \{s. \ C \ s\}$   
 and *ffb-Q*:  $P \leq fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ Q$   
 shows  $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ Q$   
**proof** (*subst ffb-eq, subst g-orbital-eq, clarsimp*)  
 fix  $t :: \text{real}$  **and**  $X :: \text{real} \Rightarrow 'a$  **and**  $s$  **assume**  $s \in P$  **and**  $t \in (U \ s)$   
 and  $x\text{-ivp}: X \in \text{Sols } f \ U \ S \ t_0 \ s$



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

**lemma** *diff-inv-axiom1*:

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

**lemma** *diff-inv-axiom2*:

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

**lemma** *diff-inv-rule*:

**assumes**  $P \leq \{s. I\ s\}$  **and** *diff-invariant I*  $f\ U\ S\ t_0\ G$  **and**  $\{s. I\ s\} \leq Q$   
**shows**  $P \leq \text{fb}_{\mathcal{F}}\ (x' = f \ \&\ G\ \text{on}\ U\ S\ @\ t_0)\ Q$   
**apply**(*rule ffb-g-orbital-inv*[*OF assms(1) - assms(3)*])  
**unfolding** *ffb-diff-inv* **using** *assms(2)* .

**end**

### 0.5.4 Examples

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

**theory** *HS-VC-PT-Examples*  
**imports** *HS-VC-PT*

**begin**

#### Pendulum

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

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

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

— Verified by providing the dynamics

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

— Verified with differential invariants.

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

— Verified with the flow.

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

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

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

#### Bouncing Ball

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

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

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

— Verified with differential invariants.

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

**lemma** *inv-imp-pos-le*[*bb-real-arith*]:

**assumes**  $0 > g$  **and** *inv*:  $2 * g * x - 2 * g * h = v * v$

**shows**  $(x::real) \leq h$

**proof**—

**have**  $v * v = 2 * g * x - 2 * g * h \wedge 0 > g$

**using** *inv* **and**  $\langle 0 > g \rangle$  **by** *auto*

**hence** *obs*:  $v * v = 2 * g * (x - h) \wedge 0 > g \wedge v * v \geq 0$

**using** *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)

**hence**  $(v * v) / (2 * g) = (x - h)$

**by** *auto*

**also from** *obs* **have**  $(v * v) / (2 * g) \leq 0$

**using** *divide-nonneg-neg* **by** *fastforce*

**ultimately have**  $h - x \geq 0$

**by** *linarith*

**thus** *?thesis* **by** *auto*

**qed**

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

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

(*LOOP* (

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

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

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

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

**apply**(*rule ffb-loopI, simp-all*)

**apply**(*force, force simp: bb-real-arith*)

**apply**(*rule ffb-g-odei*)

**by** (*auto intro!: diff-invariant-rules poly-derivatives simp: bb-real-arith*)

— Verified by providing the dynamics

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

**assumes** *invar*:  $2 * g * x = 2 * g * h + v * v$

**and** *pos*:  $g * \tau^2 / 2 + v * \tau + (x::real) = 0$

**shows**  $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$

**proof**—

**from** *pos* **have**  $g * \tau^2 + 2 * v * \tau + 2 * x = 0$  **by** *auto*

**then have**  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x = 0$

**by** (*metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right*

*monoid-mult-class.power2-eq-square semiring-class.distrib-left*)

**hence**  $g^2 * \tau^2 + 2 * g * v * \tau + v^2 + 2 * g * h = 0$

**using** *invar* **by** (*simp add: monoid-mult-class.power2-eq-square*)

**hence** *obs*:  $(g * \tau + v)^2 + 2 * g * h = 0$

**apply**(*subst power2-sum*) **by** (*metis (no-types, hide-lams) Groups.add-ac(2, 3)*

*Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2)*)

**thus**  $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$

**by** (*simp add: add.commute distrib-right power2-eq-square*)

**qed**

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

**assumes** *invar*:  $2 * g * x = 2 * g * h + v * v$

**shows**  $2 * g * (g * \tau^2 / 2 + v * \tau + (x::real)) =$

$2 * g * h + (g * \tau + v) * (g * \tau + v)$  (**is** *?lhs = ?rhs*)

**proof**—

**have** *?lhs*  $= g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$

**by**(*auto simp: algebra-simps semiring-normalization-rules(29)*)

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

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

— Verified with the flow.

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

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

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

## Thermostat

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

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

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

— Verified with the flow.

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

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

**assume** *a1*:  $0 < a$

**have** *f2*:  $\bigwedge r\ ra. |(r::real) + -\ ra| = |ra + -\ r|$

**by** (*metis abs-minus-commute minus-real-def*)

**have**  $\bigwedge r\ ra\ rb. (r::real) * ra + -\ (r * rb) = r * (ra + -\ rb)$

**by** (*metis minus-real-def right-diff-distrib*)

**hence**  $|a * (s_1\$1 + -\ L) + -\ (a * (s_2\$1 + -\ L))| = a * |s_1\$1 + -\ s_2\$1|$

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

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

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

**qed**

**lemma** *local-lipschitz-temp-dyn*:

**assumes**  $0 < (a::real)$

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

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

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

**using** *assms*

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

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

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

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

**by** (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-temp-dyn simp: forall-4 vec-eq-iff*)

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

**assumes**  $a > 0$  **and** *Thyps*:  $0 < Tmin\ Tmin \leq T\ T \leq Tmax$

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

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

**proof**–

**have**  $0 \leq t \wedge t \leq -(\ln(Tmin / T) / a)$

**using** *thyys* **by** *auto*

**hence**  $\ln(Tmin / T) \leq -a * t \wedge -a * t \leq 0$

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

**also have**  $Tmin / T > 0$

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

**ultimately have** *obs*:  $Tmin / T \leq \exp(-a * t)\ \exp(-a * t) \leq 1$

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

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

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

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

**using** *Thyps* *mult-left-le-one-le[OF - exp-ge-zero obs(2), of T]*

*less-eq-real-def order-trans-rules(23)* **by** *blast*

**qed**

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

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

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

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

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

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

**proof**–

**have**  $0 \leq t \wedge t \leq -(\ln((L - Tmax) / (L - T)) / a)$

**using** *thyys* **by** *auto*

**hence**  $\ln((L - Tmax) / (L - T)) \leq -a * t \wedge -a * t \leq 0$

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

**also have**  $(L - Tmax) / (L - T) > 0$

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

```

ultimately have  $(L - Tmax) / (L - T) \leq \exp(-a * t) \wedge \exp(-a * t) \leq 1$ 
  using exp-ln exp-le-one-iff by (metis exp-less-cancel-iff not-less)
moreover have  $L - T > 0$ 
  using Thyps by auto
ultimately have obs:  $(L - Tmax) \leq \exp(-a * t) * (L - T) \wedge \exp(-a * t) * (L - T) \leq (L - T)$ 
  by (simp add: pos-divide-le-eq)
thus  $(L - Tmax) \leq \exp(-(a * t)) * (L - T)$ 
  by auto
thus  $L - \exp(-(a * t)) * (L - T) \leq Tmax$ 
  by auto
show  $Tmin \leq L - \exp(-(a * t)) * (L - T)$ 
  using Thyps and obs by auto
qed

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

lemma thermostat:
  assumes  $a > 0$  and  $0 \leq t$  and  $0 < Tmin$  and  $Tmax < L$ 
  shows  $\{s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0\} \leq fb_{\mathcal{F}}$ 
  (LOOP
    — control
    (( $2 ::= (\lambda s. 0)$ );( $3 ::= (\lambda s. s\$1)$ ));
    (IF  $(\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1)$  THEN  $(4 ::= (\lambda s. 1))$  ELSE
    (IF  $(\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1)$  THEN  $(4 ::= (\lambda s. 0))$  ELSE skip));
    — dynamics
    (IF  $(\lambda s. s\$4 = 0)$  THEN  $(x' = (\lambda t. f\ a\ 0) \ \& \ (\lambda s. s\$2 \leq -(\ln(Tmin/s\$3))/a)$  on  $(\lambda s. \{0..t\})$  UNIV
    @ 0)
    ELSE  $(x' = (\lambda t. f\ a\ L) \ \& \ (\lambda s. s\$2 \leq -(\ln((L-Tmax)/(L-s\$3))/a)$  on  $(\lambda s. \{0..t\})$  UNIV @ 0)) )
    INV  $(\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1))$ 
     $\{s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax\}$ 
    apply(rule ffb-loopI, simp-all add: ffb-temp-dyn[OF assms(1,2)] le-fun-def, safe)
    using temp-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin]
    and temp-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

no-notation temp-vec-field (f)
  and temp-flow ( $\varphi$ )
end

```

## 0.6 Verification components with MKA

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

```

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

```

```
begin
```

### 0.6.1 Verification in AKA

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

```

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

```

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

**context** *antidomain-kleene-algebra*  
**begin**

— Skip

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

— Abort

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

— Sequential composition

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

**declare** *fbox-mult* [*simp*]

— Nondeterministic choice

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

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

— Conditional statement

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

**lemma** *fbox-export1*:  $ad \ p + |x] \ q = |d \ p \cdot x] \ q$   
**using** *a-d-add-closure addual.ars-r-def fbox-def fbox-mult* **by** *auto*

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

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

**have**  $obs: ?lhs = d \ p \cdot ?lhs + ad \ p \cdot ?lhs$   
**by** (*metis (no-types, lifting) local.a-closure' local.a-de-morgan fbox-def ans-d-def ads-d-def local.am2 local.am5-lem local.dka.dsg3 local.dka.dsr5*)

**have**  $d \ p \cdot ?lhs = d \ p \cdot |x] \ q \cdot (d \ p + d \ (|y] \ q))$   
**using** *fbox-cond local.a-d-add-closure local.ads-d-def local.ds.ddual.mult-assoc local.fbox-def* **by** *auto*

**also have**  $\dots = d \ p \cdot |x] \ q$   
**by** (*metis local.ads-d-def local.am2 local.dka.dns5 local.ds.ddual.mult-assoc local.fbox-def*)

**finally have**  $d \ p \cdot ?lhs = d \ p \cdot |x] \ q$  .

**moreover have**  $ad \ p \cdot ?lhs = ad \ p \cdot |y] \ q$   
**by** (*metis add-commute fbox-cond local.a-closure' local.a-mult-add ads-d-def ans-d-def local.dnsz.dns5 local.ds.ddual.mult-assoc local.fbox-def*)

**ultimately show** *?thesis*

**using** *obs* **by** *simp*

**qed**

— While loop

**definition** *aka-whilei* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a (*while* - *do* - *inv* - [64,64,64] 63) **where**  
*while* *t do x inv i* = (*d t* · *x*)<sup>\*</sup> · *ad t*

**lemma** *fbox-frame*:  $d p \cdot x \leq x \cdot d p \implies d q \leq |x| \ r \implies d p \cdot d q \leq |x| \ (d p \cdot d r)$   
**using** *dual.mult-isol-var fbox-add1 fbox-demodalisation3 fbox-simp* **by** *auto*

**lemma** *fbox-shunt*:  $d p \cdot d q \leq |x| \ t \iff d p \leq ad \ q + |x| \ t$   
**by** (*metis a-6 a-antitone' a-loc add-commute addual.ars-r-def am-d-def da-shunt2 fbox-def*)

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

**proof** –

{**fix** *t*

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

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

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

**by** (*metis a-closure' addual.ars-r-def ans-d-def dka.dsg3 ds.ddual.mult-assoc fbox-def fbox-demodalisation3*)}

**thus** *?thesis*

**by** (*metis a-closure' addual.ars-r-def ans-d-def fbox-def order-refl*)

**qed**

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

**proof** –

**assume**  $d p \cdot d t \leq |x| \ p$

**hence**  $d p \leq |d t \cdot x| \ p$

**by** (*simp add: fbox-export1 fbox-shunt*)

**hence**  $d p \leq |(d t \cdot x)^*| \ p$

**by** (*simp add: fbox-star-induct-var*)

**thus** *?thesis*

**using** *order-trans fbox-export2* **by** *presburger*

**qed**

**lemma** *fbox-whilei*:

**assumes**  $d p \leq d i$  **and**  $d i \cdot ad \ t \leq d q$  **and**  $d i \cdot d t \leq |x| \ i$

**shows**  $d p \leq |while \ t \ do \ x \ inv \ i| \ q$

**proof**–

**have**  $d i \leq |(d t \cdot x)^* \cdot ad \ t| \ (d i \cdot ad \ t)$

**using** *fbox-while assms* **by** *blast*

**also have**  $\dots \leq |(d t \cdot x)^* \cdot ad \ t| \ q$

**by** (*metis assms(2) local.dka.dom-iso local.dka.domain-invol local.fbox-iso*)

**finally show** *?thesis*

**unfolding** *aka-whilei-def*

**using** *assms(1) local.dual-order.trans* **by** *blast*

**qed**

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

**proof** –

**assume**  $h1: p \leq |x| \ p'$  **and**  $h2: p' \leq |y| \ q$

**hence**  $|x| \ p' \leq |x| \ |y| \ q$

**by** (*metis ads-d-def fbox-antitone-var fbox-dom fbox-iso*)

**thus** *?thesis*

**by** (*metis dual-order.trans fbox-mult h1*)

**qed**

**lemma** *fbox-whilei-break*:

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

**apply** (*rule fbox-seq-var[OF - fbox-whilei]*)

**using** *fbox-simp* **by** *auto*

— Finite iteration



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

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

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

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

— Invariants

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

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

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

**oops**

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

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

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

**end**

**end**

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

**begin**

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

## 0.6.2 Relational model

**context** *dioid-one-zero*  
**begin**

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

**lemma** *power-inductr*: *z* + *y* · *x*  $\leq y$   $\Longrightarrow$  *z* · (*x*  $\wedge$  *n*)  $\leq y$

```

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

```

**end**

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

**lemma** power-is-relpow: rel-diod.power  $X\ n = X^n$

```

proof (induct n)
  case 0 show ?case
    by (metis rel-diod.power-0 relpow.simps(1))
  case Suc thus ?case
    by (metis rel-diod.power-Suc2 relpow.simps(2))
qed

```

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

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

**lemma** rel-star-contr:  $X^* O Y = (\bigcup n. (\text{rel-diod.power } X\ n) O Y)$   
**by** (metis rel-star-def relcomp-UNION-distrib2)

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

**proof**

```

  fix x y z :: 'a rel
  show  $Id \cup x O x^* \subseteq x^*$ 
    by (metis order-refl r-comp-rtrancl-eq rtrancl-unfold)
next
  fix x y z :: 'a rel
  assume  $z \cup x O y \subseteq y$ 
  thus  $x^* O z \subseteq y$ 
    by (simp only: rel-star-contr, metis (lifting) SUP-le-iff rel-diod.power-inductl)
next
  fix x y z :: 'a rel
  assume  $z \cup y O x \subseteq y$ 
  thus  $z O x^* \subseteq y$ 
    by (simp only: rel-star-contrl, metis (lifting) SUP-le-iff rel-diod.power-inductr)
qed

```

**definition**  $rel\text{-}ad :: 'a\ rel \Rightarrow 'a\ rel$  **where**  
 $rel\text{-}ad\ R = \{(x,x) \mid x. \neg (\exists y. (x,y) \in R)\}$

**interpretation**  $rel\text{-}aka$ : *antidomain-kleene-algebra*  $rel\text{-}ad\ (\cup)\ (O)\ Id\ \{\}\ (\subseteq)\ (\subset)\ rtranc1$   
**by** *unfold-locales (auto simp: rel-ad-def)*

### 0.6.3 Store and weakest preconditions

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

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

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

**definition**  $p2r :: 'a\ pred \Rightarrow 'a\ rel\ ((\lceil \cdot \rceil))$  **where**  
 $\lceil P \rceil = \{(s,s) \mid s. P\ s\}$

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

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

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

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

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

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

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

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

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

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

**lemma**  $le\text{-}wp\text{-}choice\text{-}iff$ :  $\lceil P \rceil \leq wp\ (X \cup Y)\ \lceil Q \rceil \longleftrightarrow \lceil P \rceil \leq wp\ X\ \lceil Q \rceil \wedge \lceil P \rceil \leq wp\ Y\ \lceil Q \rceil$

**using** *rel-aka.le-fbox-choice-iff*[of  $\lceil P \rceil$ ] **by** *simp*

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

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

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

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

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

### 0.6.4 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

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

**context** *local-flow*  
**begin**

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

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

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

$\lceil \lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G(\varphi \tau s)) \longrightarrow Q(\varphi t s)) \rceil$   
**apply**(subst wp-g-ode-subset, simp-all add: subintervalI init-time real-Icc-closed-segment)  
**by** (auto simp: closed-segment-eq-real-ivl)

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

**end**

Verification with differential invariants

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

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

**lemma** wp-g-orbital-inv:  
**assumes**  $\lceil P \rceil \leq \lceil I \rceil$  **and**  $\lceil I \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil I \rceil$  **and**  $\lceil I \rceil \leq \lceil Q \rceil$   
**shows**  $\lceil P \rceil \leq wp(x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$   
**using** assms(1)  
**apply**(rule order.trans)  
**using** assms(2)  
**apply**(rule order.trans)  
**apply**(rule rel-aka.fbox-iso)  
**using** assms(3) **by** auto

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

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

**context** local-flow  
**begin**

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

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

**end**

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

**unfolding** *g-ode-inv-def*  
**apply**(*rule-tac*  $b=wp\ (x'=f \ \&\ G\ on\ U\ S\ @\ t_0)\ [I]$  **in** *order.trans*)  
**apply**(*rule-tac*  $I=I$  **in** *wp-g-orbital-inv, simp-all*)  
**apply**(*subst wp-g-orbital-guard, simp*)  
**by** (*rule rel-aka.fbox-iso, simp*)

### 0.6.5 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

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

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

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

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

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

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

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

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

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

**proof**  
**fix**  $\tau$  **assume**  $\tau \in (down\ (U\ s)\ t)$   
**hence**  $x\ \tau \in g\text{-orbital}\ f\ G\ U\ S\ t_0\ s$   
**using** *assms(2)* **unfolding** *g-ode-def* **by** *blast*  
**also have**  $\forall y. y \in (g\text{-orbital}\ f\ G\ U\ S\ t_0\ s) \longrightarrow C\ y$   
**using** *assms(1)* **unfolding** *wp-rel g-ode-def* **by**(*auto simp: p2r-def*)  
**ultimately show**  $C\ (x\ \tau)$   
**by** *blast*

qed

lemma *diff-cut-axiom*:

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

lemma *diff-cut-rule*:

assumes  $wp\text{-}C: [P] \leq wp\ (x' = f \ \&\ G\ on\ U\ S\ @\ t_0)\ [C]$   
 and  $wp\text{-}Q: [P] \leq wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ U\ S\ @\ t_0)\ [Q]$   
 shows  $[P] \leq wp\ (x' = f \ \&\ G\ on\ U\ S\ @\ t_0)\ [Q]$   
 proof(*subst wp-rel, simp add: g-orbital-eq p2r-def g-ode-def, clarsimp*)  
 fix  $t :: real$  and  $X :: real \Rightarrow 'a$  and  $s$  assume  $P\ s$  and  $t \in U\ s$   
 and  $x\text{-ivp}: X \in Sols\ f\ U\ S\ t_0\ s$   
 and  $guard\text{-}x: \forall x. x \in U\ s \wedge x \leq t \longrightarrow G\ (X\ x)$   
 have  $\forall t \in (down\ (U\ s)\ t). X\ t \in g\text{-orbital}\ f\ G\ U\ S\ t_0\ s$   
 using *g-orbitalI*[*OF*  $x\text{-ivp}$ ] *guard-x* by *auto*  
 hence  $\forall t \in (down\ (U\ s)\ t). C\ (X\ t)$   
 using  $wp\text{-}C\ \langle P\ s \rangle$  by (*subst (asm) wp-rel, auto simp: g-ode-def*)  
 hence  $X\ t \in g\text{-orbital}\ f\ (\lambda s. G\ s \wedge C\ s)\ U\ S\ t_0\ s$   
 using *guard-x*  $\langle t \in U\ s \rangle$  by (*auto intro!: g-orbitalI x-ivp*)  
 thus  $Q\ (X\ t)$   
 using  $\langle P\ s \rangle\ wp\text{-}Q$  by (*subst (asm) wp-rel*) (*auto simp: g-ode-def*)  
 qed

lemma *diff-inv-axiom1*:

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

lemma *diff-inv-axiom2*:

assumes *picard-lindelof*  $(\lambda t. f)\ UNIV\ UNIV\ 0$   
 and  $\bigwedge s. \{t :: real. t \geq 0\} \subseteq \text{picard-lindelof.ex-ivl}\ (\lambda t. f)\ UNIV\ UNIV\ 0\ s$   
 and *diff-invariant*  $I\ (\lambda t. f)\ (\lambda s. \{t :: real. t \geq 0\})\ UNIV\ 0\ G$   
 shows  $wp\ (x' = f \ \&\ G)\ [I] = wp\ [G]\ [I]$   
 proof(*unfold wp-g-orbital, subst wp-rel, clarsimp simp: fun-eq-iff*)  
 fix  $s$   
 let  $?ex\text{-ivl}\ s = \text{picard-lindelof.ex-ivl}\ (\lambda t. f)\ UNIV\ UNIV\ 0\ s$   
 let  $?lhs\ s =$

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

**lemma** *diff-inv-rule*:

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

**end**

### 0.6.6 Examples

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

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

**begin**

#### Pendulum

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

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

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

— Verified by providing dynamics.

**lemma** *pendulum-dyn*:

$\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp (EVOL \varphi G T) \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$   
**by** *simp*



— Verified with differential invariants.

**lemma** *pendulum-inv*:

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

— Verified with the flow.

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

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

**lemma** *pendulum-flow*:

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

**no-notation** *fpend* (*f*)

**and** *pend-flow* ( *$\varphi$* )

## Bouncing Ball

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

**abbreviation** *fball* :: *real*  $\Rightarrow$  *real*<sup>2</sup>  $\Rightarrow$  *real*<sup>2</sup> (*f*)

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

**abbreviation** *ball-flow* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real*<sup>2</sup>  $\Rightarrow$  *real*<sup>2</sup> ( *$\varphi$* )

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

— Verified with differential invariants.

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

**lemma** *inv-imp-pos-le[bb-real-arith]*:

**assumes**  $0 > g$  **and** *inv*:  $2 * g * x - 2 * g * h = v * v$

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

**proof**—

**have**  $v * v = 2 * g * x - 2 * g * h \wedge 0 > g$

**using** *inv* **and**  $\langle 0 > g \rangle$  **by** *auto*

**hence** *obs*:  $v * v = 2 * g * (x - h) \wedge 0 > g \wedge v * v \geq 0$

**using** *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)

**hence**  $(v * v) / (2 * g) = (x - h)$

**by** *auto*

**also from** *obs* **have**  $(v * v) / (2 * g) \leq 0$

**using** *divide-nonneg-neg* **by** *fastforce*

**ultimately have**  $h - x \geq 0$

**by** *linarith*

**thus** *?thesis* **by** *auto*

**qed**

**lemma** *bouncing-ball-inv*:

**fixes** *h* :: *real*

**shows**  $g < 0 \implies h \geq 0 \implies \lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq$

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

— Verified by providing dynamics.

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

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

proof—

from pos have  $g * \tau^2 + 2 * v * \tau + 2 * x = 0$  by auto  
 then have  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x = 0$   
 by (metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right  
 monoid-mult-class.power2-eq-square semiring-class.distrib-left)  
 hence  $g^2 * \tau^2 + 2 * g * v * \tau + v^2 + 2 * g * h = 0$   
 using invar by (simp add: monoid-mult-class.power2-eq-square)  
 hence obs:  $(g * \tau + v)^2 + 2 * g * h = 0$   
 apply(subst power2-sum) by (metis (no-types, hide-lams) Groups.add-ac(2, 3)  
 Groups.mult-ac(2, 3) monoid-mult-class.power2-eq-square nat-distrib(2))  
 thus  $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$   
 by (simp add: monoid-mult-class.power2-eq-square)

qed

lemma inv-conserv-at-air[bb-real-arith]:

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

proof—

have ?lhs =  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$   
 by(auto simp: algebra-simps semiring-normalization-rules(29))  
 also have ... =  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * h + v * v$  (is ... = ?middle)  
 by(subst invar, simp)  
 finally have ?lhs = ?middle.  
 moreover  
 {have ?rhs =  $g * g * (\tau * \tau) + 2 * g * v * \tau + 2 * g * h + v * v$   
 by (simp add: Groups.mult-ac(2,3) semiring-class.distrib-left)  
 also have ... = ?middle  
 by (simp add: semiring-normalization-rules(29))  
 finally have ?rhs = ?middle.}  
 ultimately show ?thesis by auto

qed

lemma bouncing-ball-dyn:

fixes h::real  
 assumes  $g < 0$  and  $h \geq 0$   
 shows  $g < 0 \implies h \geq 0 \implies$   
 $\lceil \lambda\ s. \ s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$   
 (LOOP  
 ((EVOL  $(\varphi\ g) (\lambda\ s. \ 0 \leq s\$1) T$ );  
 (IF  $(\lambda\ s. \ s\$1 = 0)$  THEN  $(2 ::= (\lambda\ s. - s\$2))$  ELSE skip))  
 INV  $(\lambda\ s. \ 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)$   
 $\lceil \lambda\ s. \ 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
 by (rule wp-loopI) (auto simp: bb-real-arith)

— Verified with the flow.

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

**lemma** *bouncing-ball-flow*:  
**fixes** *h::real*  
**assumes**  $g < 0$  **and**  $h \geq 0$   
**shows**  $g < 0 \implies h \geq 0 \implies$   
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$   
 (*LOOP*  
 ( $(x' = (\lambda t. f\ g) \ \& \ (\lambda s. s\$1 \geq 0))$  *on* ( $\lambda s. UNIV$ ) *UNIV* @ 0);  
 (*IF* ( $\lambda s. s\$1 = 0$ ) *THEN* ( $2 ::= (\lambda s. -\ s\$2)$ ) *ELSE skip*))  
*INV* ( $\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$ ))  
 $\lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$ )  
**apply**(*rule wp-loopI*, *simp-all* add: *local-flow.wp-g-ode-subset[OF local-flow-ball]*)  
**by** (*auto simp: bb-real-arith*)

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

## Thermostat

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

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

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

— Verified with the flow.

**lemma** *norm-diff-temp-dyn*:  $0 < a \implies \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$1 - s_2\$1|$   
**proof**(*simp* add: *norm-vec-def L2-set-def*, *unfold UNIV-4*, *simp*)  
**assume**  $a1: 0 < a$   
**have**  $f2: \bigwedge r\ ra. |(r::real) + -\ ra| = |ra + -\ r|$   
**by** (*metis abs-minus-commute minus-real-def*)  
**have**  $\bigwedge r\ ra\ rb. (r::real) * ra + -\ (r * rb) = r * (ra + -\ rb)$   
**by** (*metis minus-real-def right-diff-distrib*)  
**hence**  $|a * (s_1\$1 + -\ L) + -\ (a * (s_2\$1 + -\ L))| = a * |s_1\$1 + -\ s_2\$1|$   
**using** *a1* **by** (*simp* add: *abs-mult*)  
**thus**  $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$   
**using** *f2 minus-real-def* **by** *presburger*  
**qed**

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

```

apply(unfold local-lipschitz-def lipschitz-on-def dist-norm)
apply(clarsimp, rule-tac x=1 in exI, clarsimp, rule-tac x=a in exI)
using assms
apply(simp-all add: norm-diff-temp-dyn)
apply(simp add: norm-vec-def L2-set-def, unfold UNIV-4, clarsimp)
unfolding real-sqrt-abs[symmetric] by (rule real-le-lsqrt) auto

```

**lemma** *local-flow-temp*:  $a > 0 \implies \text{local-flow } (f \ a \ L) \text{ UNIV UNIV } (\varphi \ a \ L)$   
**by** (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-temp-dyn simp: forall-4 vec-eq-iff*)

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

```

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

```

**proof**–

```

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

```

**qed**

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

```

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

```

**proof**–

```

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

```

**qed**

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

**lemma** *thermostat*:

```

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

no-notation temp-vec-field (f)
  and temp-flow ( $\varphi$ )

end

```

## 0.7 Verification components with MKA and non-deterministic functions

We show that non-deterministic endofunctions form an antidomain Kleene algebra (hence a modal Kleene algebra). We use MKA's forward box operator to derive rules for weakest liberal preconditions (wlps) of hybrid programs. Finally, we derive our three methods for verifying correctness specifications for the continuous dynamics of HS.

```

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

```

```
begin
```

### 0.7.1 Non-deterministic functions

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

```

notation Abs-nd-fun ( $-\bullet$  [101] 100)
  and Rep-nd-fun ( $-\bullet$  [101] 100)
  and fbox (wp)

```

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

```

lemma nd-fun-ext:  $(\bigwedge x. (f\bullet) x = (g\bullet) x) \implies f = g$ 
apply(subgoal-tac Rep-nd-fun f = Rep-nd-fun g)
using Rep-nd-fun-inject
apply blast
by(rule ext, simp)

```

```

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

```

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

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

**definition**  $0 = \zeta^\bullet$

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

**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**  $x::'a\ nd\text{-}fun$   
**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**  $x::'a\ nd\text{-}fun$   
**unfolding** *plus-nd-fun-def times-nd-fun-def* **by** (*simp-all add: kcomp-distr kcomp-distl*)

**lemma** *nd-fun-plus-zero*[*nd-fun-aka*]:  $0 + x = x$   
**and** *nd-fun-mult-zero*[*nd-fun-aka*]:  $0 \cdot x = 0$   
**and** *nd-fun-mult-zero*[*nd-fun-aka*]:  $x \cdot 0 = 0$  **for**  $x::'a\ nd\text{-}fun$   
**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**  $x::'a\ nd\text{-}fun$   
**unfolding** *less-eq-nd-fun-def less-nd-fun-def plus-nd-fun-def times-nd-fun-def sup-fun-def*  
**by** (*unfold nd-fun-eq-iff le-fun-def, auto simp: kcomp-def*)

**lemma** *nd-fun-ad-zero*[*nd-fun-aka*]:  $ad\ x \cdot x = 0$   
**and** *nd-fun-ad*[*nd-fun-aka*]:  $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$   
**and** *nd-fun-ad-one*[*nd-fun-aka*]:  $ad\ (ad\ x) + ad\ x = 1$  **for**  $x::'a\ nd\text{-}fun$   
**unfolding** *antidomain-op-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def*  
**by** (*auto simp: nd-fun-eq-iff kcomp-def one-nd-fun-def*)

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

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

**end**

## 0.7.2 Store and weakest preconditions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**lemma**  $wp\text{-loopI-break}$ :  
 $\lceil P \rceil \leq wp \ Y \ \lceil I \rceil \Longrightarrow \lceil I \rceil \leq wp \ X \ \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \lceil Q \rceil \Longrightarrow \lceil P \rceil \leq wp \ (Y ; (LOOP \ X \ INV \ I)) \ \lceil Q \rceil$

using *fbx-loopi-break*[of  $\lceil P \rceil$ ] by *auto*

### 0.7.3 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

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

**context** *local-flow*  
**begin**

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

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

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

**lemma** *wp-orbit*:  $wp (\gamma^\varphi)^\bullet \lceil Q \rceil = \lceil \lambda s. s \in S \longrightarrow (\forall t \in T. Q (\varphi t s)) \rceil$   
 unfolding *orbit-def wp-g-ode g-ode-def[symmetric]* by *auto*

**end**

Verification with differential invariants

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

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



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

**lemma** *wp-g-orbital-inv*:

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

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

**using** *assms*(1)

**apply**(*rule order.trans*)

**using** *assms*(2)

**apply**(*rule order.trans*)

**apply**(*rule fbox-iso*)

**using** *assms*(3) **by** *auto*

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

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

**lemma** *diff-inv-guard-ignore*:

**assumes**  $\lceil I \rceil \leq \text{wp } (x' = f \ \& \ (\lambda s. \text{True}) \text{ on } U \ S \ @ \ t_0) \ \lceil I \rceil$

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

**using** *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

**context** *local-flow*

**begin**

**lemma** *wp-diff-inv-eq*:

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

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

$(\lceil \lambda s. s \in S \implies I \ s \rceil = \text{wp } (x' = (\lambda t. f) \ \& \ (\lambda s. \text{True}) \text{ on } U \ S \ @ \ 0) \ \lceil \lambda s. s \in S \implies I \ s \rceil)$

**unfolding** *wp-diff-inv[symmetric]*

**apply**(*subst wp-g-ode-subset[OF assms]*, *simp*)**+**

**apply**(*clarsimp*, *safe*, *force*)

**apply**(*erule-tac x=0 in ballE*)

**using** *init-time in-domain ivp*(2) *assms* **apply**(*force*, *force*)

**apply**(*erule-tac x=s in allE*, *clarsimp*, *erule-tac x=t in ballE*)

**using** *in-domain ivp*(2) *assms* **by** *force***+**

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

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

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

**end**

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

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

**unfolding** *g-ode-inv-def*

**apply**(*rule-tac b=wp* ( $x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0$ )  $\lceil I \rceil$  **in** *order.trans*)

**apply**(*rule-tac I=I* **in** *wp-g-orbital-inv*, *simp-all*)

**apply**(*subst wp-g-orbital-guard*, *simp*)

**by** (*rule fbox-iso*, *simp*)

#### 0.7.4 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL). First we present a generalised version, then we show the rules as instances of the general ones.

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

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

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

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

**lemma** *diff-solve-axiom1*:

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

**lemma** *diff-solve-axiom2*:

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

**lemma** *diff-solve-rule*:

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

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

**unfolding** *wp-nd-fun g-ode-def g-orbital-eq less-eq-nd-fun-def*  
**by** (*auto simp: le-fun-def*)

**lemma** *diff-weak-axiom2*:

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

**lemma** *diff-weak-rule*:

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

**lemma** *wp-g-orbit-IdD*:

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

**proof**

**fix**  $\tau$  **assume**  $\tau \in (\text{down } (U \ s) \ t)$   
**hence**  $x \ \tau \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$   
**using** *assms(2)* **by** *blast*  
**also have**  $\forall y. y \in (g\text{-orbital } f \ G \ U \ S \ t_0 \ s) \longrightarrow C \ y$   
**using** *assms(1)* **unfolding** *wp-nd-fun g-ode-def*  
**by** (*subst (asm) nd-fun-eq-iff*) *auto*  
**ultimately show**  $C \ (x \ \tau)$   
**by** *blast*

**qed**

**lemma** *diff-cut-axiom*:

**assumes**  $\text{wp } (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [C] = \eta^\bullet$   
**shows**  $\text{wp } (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \ [Q] = \text{wp } (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \ [Q]$

**proof** (*rule-tac  $f = \lambda x. \text{wp } x \ [Q]$  in HOL.arg-cong, rule nd-fun-ext, rule subset-antisym*)

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

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

**fix**  $s'$  **assume**  $s' \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$

**then obtain**  $\tau::\text{real}$  **and**  $X$  **where**  $x\text{-ivp: } X \in \text{ivp-sols } f \ U \ S \ t_0 \ s$

**and**  $X \ \tau = s'$  **and**  $\tau \in U \ s$  **and**  $\text{guard-}x: (\mathcal{P} \ X \ (\text{down } (U \ s) \ \tau) \subseteq \{s. G \ s\})$

**using** *g-orbitalD[of  $s' f G - S \ t_0 \ s$ ]* **by** *blast*

**have**  $\forall t \in (\text{down } (U \ s) \ \tau). \mathcal{P} \ X \ (\text{down } (U \ s) \ t) \subseteq \{s. G \ s\}$

**using** *guard-x* **by** (*force simp: image-def*)

**also have**  $\forall t \in (\text{down } (U \ s) \ \tau). t \in (U \ s)$

using  $\langle \tau \in (U\ s) \rangle$  by *auto*  
 ultimately have  $\forall t \in (\text{down } (U\ s)\ \tau). X\ t \in g\text{-orbital } f\ G\ U\ S\ t_0\ s$   
 using *g-orbitalI[OF x-ivp]* by (*metis (mono-tags, lifting)*)  
 hence  $\forall t \in (\text{down } (U\ s)\ \tau). C\ (X\ t)$   
 using *wp-g-orbit-IdD[OF assms(1)]* by *blast*  
 thus  $s' \in g\text{-orbital } f\ (\lambda s. G\ s \wedge C\ s)\ U\ S\ t_0\ s$   
 using *g-orbitalI[OF x-ivp \(\tau \in (U\ s)\)] guard-x \(\langle X\ \tau = s' \rangle* by *fastforce*  
 qed  
 next  
 fix  $s$  show  $((x' = f \ \& \ \lambda s. G\ s \wedge C\ s \text{ on } U\ S\ @\ t_0)_{\bullet})\ s \subseteq ((x' = f \ \& \ G \text{ on } U\ S\ @\ t_0)_{\bullet})\ s$   
 by (*auto simp: g-orbital-eq g-ode-def*)  
 qed

lemma *diff-cut-rule*:

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

lemma *diff-inv-axiom1*:

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

lemma *diff-inv-axiom2*:

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

```

moreover
{assume  $G\ s \longrightarrow I\ s$ 
  hence ?lhs  $s$ 
    apply(clarify, subgoal-tac  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$ )
    apply(erule-tac  $x=0$  in  $allE$ , frule ivp-solsD(2), simp)
    using  $dinv$  by blast+}
ultimately show ?lhs  $s = (G\ s \longrightarrow I\ s)$ 
  by blast
qed

```

**lemma** *diff-inv-rule*:

```

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

```

**end**

### 0.7.5 Examples

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

```

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

```

**begin**

#### Pendulum

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

```

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

```

```

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

```

— Verified by providing dynamics.

**lemma** *pendulum-dyn*:

```

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

```

— Verified with differential invariants.

**lemma** *pendulum-inv*:

```

 $\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp\ (x' = f \ \&\ G)\ \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$ 
by (auto intro!: poly-derivatives diff-invariant-rules)

```

— Verified with the flow.

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

```

apply(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff, clarsimp)
apply(rule-tac  $x=1$  in  $exI$ , clarsimp, rule-tac  $x=1$  in  $exI$ )

```

**apply**(*simp add: dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)  
**by** (*auto simp: forall-2 intro!: poly-derivatives*)

**lemma** *pendulum-flow*:

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

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

## Bouncing Ball

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

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

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

— Verified with differential invariants.

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

**lemma** *inv-imp-pos-le[bb-real-arith]*:

**assumes**  $0 > g$  **and** *inv*:  $2 * g * x - 2 * g * h = v * v$   
**shows**  $(x :: real) \leq h$

**proof**—

**have**  $v * v = 2 * g * x - 2 * g * h \wedge 0 > g$   
**using** *inv* **and**  $\langle 0 > g \rangle$  **by** *auto*  
**hence** *obs*:  $v * v = 2 * g * (x - h) \wedge 0 > g \wedge v * v \geq 0$   
**using** *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)  
**hence**  $(v * v) / (2 * g) = (x - h)$   
**by** *auto*  
**also from** *obs* **have**  $(v * v) / (2 * g) \leq 0$   
**using** *divide-nonneg-neg* **by** *fastforce*  
**ultimately have**  $h - x \geq 0$   
**by** *linarith*  
**thus** *?thesis* **by** *auto*

**qed**

**lemma** *bouncing-ball-inv*:

**fixes**  $h :: real$   
**shows**  $g < 0 \implies h \geq 0 \implies \lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq$   
 $wp$   
 $(LOOP$   
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0) \ DINV \ (\lambda s. 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0));$   
 $(IF \ (\lambda s. s\$1 = 0) \ THEN \ (2 ::= (\lambda s. - s\$2)) \ ELSE \ skip))$   
 $INV \ (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 - 2 * g * h - s\$2 * s\$2 = 0)$   
 $) \ \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
**apply**(*rule wp-loopI, simp-all, force simp: bb-real-arith*)  
**by** (*rule wp-g-odei*) (*auto intro!: poly-derivatives diff-invariant-rules*)

— Verified by providing dynamics.

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

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

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

**lemma** *bouncing-ball-dyn*:  
**fixes** *h::real*  
**assumes**  $g < 0$  **and**  $h \geq 0$   
**shows**  $g < 0 \implies h \geq 0 \implies$   
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$   
 $(LOOP$   
 $((EVOL (\varphi g) (\lambda s. 0 \leq s\$1) T);$   
 $(IF (\lambda s. s\$1 = 0) THEN (2 ::= (\lambda s. - s\$2)) ELSE skip))$   
 $INV (\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2))$   
 $\lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
**by** (*rule wp-loopI*) (*auto simp: bb-real-arith*)

— Verified with the flow.

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

**lemma** *bouncing-ball-flow*:  
**fixes** *h::real*  
**assumes**  $g < 0$  **and**  $h \geq 0$   
**shows**  $g < 0 \implies h \geq 0 \implies$

```

 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \leq wp$ 
  (LOOP
    (( $x' = (\lambda t. f\ g) \ \& \ (\lambda s. s\$1 \geq 0)$  on  $(\lambda s. UNIV) \ UNIV \ @ \ 0$ );
    (IF  $(\lambda s. s\$1 = 0)$  THEN  $(2 ::= (\lambda s. -\ s\$2))$  ELSE skip))
    INV  $(\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2)$ 
 $\lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$ 
  )
  apply(rule wp-loopI, simp-all add: local-flow.wp-g-ode-subset[OF local-flow-ball])
  by (auto simp: bb-real-arith)

```

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

## Thermostat

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

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

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

— Verified with the flow.

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

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

```

  assume a1: 0 < a
  have f2:  $\bigwedge r\ ra. |(r::real) + -\ ra| = |ra + -\ r|$ 
    by (metis abs-minus-commute minus-real-def)
  have  $\bigwedge r\ ra\ rb. (r::real) * ra + -\ (r * rb) = r * (ra + -\ rb)$ 
    by (metis minus-real-def right-diff-distrib)
  hence  $|a * (s_1\$1 + -\ L) + -\ (a * (s_2\$1 + -\ L))| = a * |s_1\$1 + -\ s_2\$1|$ 
    using a1 by (simp add: abs-mult)
  thus  $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$ 
    using f2 minus-real-def by presburger

```

qed

**lemma** local-lipschitz-temp-dyn:

```

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

```

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

by (unfold-locales, auto intro!: poly-derivatives local-lipschitz-temp-dyn simp: forall-4 vec-eq-iff)

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

assumes  $a > 0$  and  $Thyps: 0 < Tmin \ Tmin \leq T \ T \leq Tmax$

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

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

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

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

**lemma** *thermostat*:

assumes  $a > 0$  and  $0 \leq t$  and  $0 < Tmin$  and  $Tmax < L$   
 shows  $\lceil \lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge s\$4 = 0 \rceil \leq wp$   
 (*LOOP*  
 — *control*  
 ((2 ::= ( $\lambda s. 0$ )); (3 ::= ( $\lambda s. s\$1$ )));  
 (*IF* ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1$ ) *THEN* (4 ::= ( $\lambda s. 1$ )) *ELSE*  
 (*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$ ) *THEN* (4 ::= ( $\lambda s. 0$ )) *ELSE skip*));  
 — *dynamics*  
 (*IF* ( $\lambda s. s\$4 = 0$ ) *THEN* ( $x' = (\lambda t. f a 0) \ \& \ (\lambda s. s\$2 \leq -(\ln (Tmin/s\$3))/a)$  *on* ( $\lambda s. \{0..t\}$ ) *UNIV*  
 @ 0)  
*ELSE* ( $x' = (\lambda t. f a L) \ \& \ (\lambda s. s\$2 \leq -(\ln ((L - Tmax)/(L - s\$3)))/a)$  *on* ( $\lambda s. \{0..t\}$ ) *UNIV* @ 0)) )  
*INV* ( $\lambda s. Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)$ ))



```

[ $\lambda s. T_{min} \leq s\$1 \wedge s\$1 \leq T_{max}$ ]
apply(rule wp-loopI, simp-all add: fbox-temp-dyn[OF assms(1,2)])
using temp-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin]
and temp-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

no-notation temp-vec-field (f)
and temp-flow ( $\varphi$ )

end

```

## 0.8 Verification components with KAT

We create verification rules based on various Kleene Algebras.

```

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

begin

```

### 0.8.1 Hoare logic in KAT

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

```

notation t (tt)

hide-const t

no-notation if-then-else (if - then - else - fi [64,64,64] 63)
and HOL.If ((if (-)/ then (-)/ else (-)) [0, 0, 10] 10)
and while (while - do - od [64,64] 63)

context kat
begin

```

— Definitions of Hoare Triple

```

definition Hoare :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  bool (H) where
  H p x q  $\longleftrightarrow$  tt p  $\cdot$  x  $\leq$  x  $\cdot$  tt q

```

```

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

```

```

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

```

```

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

```

— Skip

```

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

```

— Abort

```

lemma H-abort: H p 0 q
by (simp add: Hoare-def)

```

— Sequential composition

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

— Nondeterministic choice

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

— Conditional statement

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

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

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

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

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

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

**finally show** *?thesis*

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

**qed**

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

— While loop

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

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

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

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

**proof** —

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

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

**using** *n-preserve test-mult* **by** *presburger*

**then show** *?thesis*

**using** *a1 Hoare-def H-exp1 conway.phl.it-simr phl-export2 kat-while-def* **by** *auto*

**qed**

**lemma** *H-while-inv*:  $\text{tt } p \leq \text{tt } i \implies \text{tt } i \cdot n\ r \leq \text{tt } q \implies H\ (\text{tt } i \cdot \text{tt } r)\ x\ i \implies H\ p\ (\text{while } r \text{ inv } i \text{ do } x)\ q$   
**by** (*metis H-cons H-while test-mult kat-while-inv-def*)

— Finite iteration

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

**lemma** *H-star-inv*:

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

**proof**—

**have**  $H \ i \ (x^*) \ i$   
**using** *assms(2) H-star* **by** *blast*  
**hence**  $H \ p \ (x^*) \ i$   
**unfolding** *Hoare-def* **using** *assms(1) phl-cons1* **by** *blast*  
**thus** *?thesis*  
**unfolding** *Hoare-def* **using** *assms(3) phl-cons2* **by** *blast*

**qed**

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

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

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

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

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

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

— Invariants

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

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

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

**unfolding** *Hoare-def* **using** *combine-common-factor*  
**by** (*smt add-commute add.left-commute distrib-left join.sup.absorb-iff1 t-add-closed*)

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

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

**end**

## 0.8.2 refinement KAT

**class** *rkate* = *kat* +

**fixes** *Ref* ::  $'a \Rightarrow 'a \Rightarrow 'a$

**assumes** *spec-def*:  $x \leq \text{Ref } p \ q \longleftrightarrow H \ p \ x \ q$

**begin**

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

**using** *spec-def* **by** *blast*

**lemma** *R2*:  $H \ p \ x \ q \Longrightarrow x \leq \text{Ref } p \ q$

**by** (*simp add: spec-def*)

**lemma** *R-cons*:  $\text{tt } p \leq \text{tt } p' \Longrightarrow \text{tt } q' \leq \text{tt } q \Longrightarrow \text{Ref } p' \ q' \leq \text{Ref } p \ q$

**proof** —

**assume** *h1*:  $\text{tt } p \leq \text{tt } p'$  **and** *h2*:  $\text{tt } q' \leq \text{tt } q$

**have**  $H \ p' \ (\text{Ref } p' \ q') \ q'$

**by** (*simp add: R1*)

**hence**  $H \ p \ (\text{Ref } p' \ q') \ q$

**using** *h1 h2 H-consl H-consr* **by** *blast*

**thus** *?thesis*

**by** (*rule R2*)

**qed**

— Skip

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

**proof** —

**have**  $H \ p \ 1 \ p$   
     **by** (*simp add: H-skip*)  
**thus** *?thesis*  
   **by** (*rule R2*)

**qed**

— Abort

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

**proof** —

**have**  $H \ 0 \ x \ 1$   
     **by** (*simp add: Hoare-def*)  
**thus** *?thesis*  
   **by** (*rule R2*)

**qed**

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

**proof** —

**have**  $H \ 1 \ (\text{Ref } 1 \ 0) \ 0$   
     **by** (*simp add: R1*)  
**thus** *?thesis*  
   **by** (*simp add: Hoare-def join.le-bot*)

**qed**

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

**using** *bot-least* **by** *force*

— Sequential composition

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

**proof** —

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

**qed**

— Nondeterministic choice

**lemma** *R-choice*:  $(\text{Ref } p \ q) + (\text{Ref } p \ q) \leq \text{Ref } p \ q$

**unfolding** *spec-def* **by** (*rule H-choice*) (*rule R1*)  
   **+**

— Conditional statement

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

**proof** —

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

**qed**

— While loop

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

**proof** —

**have**  $H (\text{tt } p \cdot \text{tt } q) (\text{Ref } (\text{tt } p \cdot \text{tt } q) \text{ } p) \text{ } p$

**by** (*simp-all add: R1*)

**hence**  $H p (\text{while } q \text{ do } (\text{Ref } (\text{tt } p \cdot \text{tt } q) \text{ } p)) (\text{tt } p \cdot n \text{ } q)$

**by** (*simp add: H-while*)

**thus** *?thesis*

**by** (*rule R2*)

**qed**

— Finite iteration

**lemma** *R-star*:  $(\text{Ref } i \text{ } i)^* \leq \text{Ref } i \text{ } i$

**proof** —

**have**  $H i (\text{Ref } i \text{ } i) \text{ } i$

**using** *R1* **by** *blast*

**hence**  $H i ((\text{Ref } i \text{ } i)^*) \text{ } i$

**using** *H-star* **by** *blast*

**thus**  $\text{Ref } i \text{ } i^* \leq \text{Ref } i \text{ } i$

**by** (*rule R2*)

**qed**

**lemma** *R-loop*:  $\text{loop } (\text{Ref } p \text{ } p) \text{ } \text{inv } i \leq \text{Ref } p \text{ } p$

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

— Invariants

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

**using** *R-cons* **by** *force*

**end**

**end**

## 0.9 Verification and refinement of HS in the relational KAT

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

**theory** *HS-VC-KAT-rel*

**imports**

*HS-VC-KAT*

$\dots / \text{HS-ODEs}$

**begin**

— We start by deleting some conflicting notation.

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

**and** *Archimedean-Field.floor-ceiling-class.floor* ( $\lfloor \cdot \rfloor$ )

**and** *tau* ( $\tau$ )

**and** *n-op* ( $n - [90] \ 91$ )

### 0.9.1 Relational model

**context** *dioid-one-zero*  
**begin**

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

**lemma** *power-inductr*:  $z + y \cdot x \leq y \implies z \cdot (x \wedge n) \leq y$   
**proof** (*induct n*)

**case** 0 **show** ?case

**using** 0.premis **by** *auto*

**case** *Suc*

**{fix** *n*

**assume**  $z + y \cdot x \leq y \implies z \cdot x \wedge n \leq y$

**and**  $z + y \cdot x \leq y$

**hence**  $z \cdot x \wedge n \leq y$

**by** *auto*

**also have**  $z \cdot x \wedge \text{Suc } n = z \cdot x \cdot x \wedge n$

**by** (*metis mult.assoc power-Suc*)

**moreover have**  $\dots = (z \cdot x \wedge n) \cdot x$

**by** (*metis mult.assoc power-commutes*)

**moreover have**  $\dots \leq y \cdot x$

**by** (*metis calculation(1) mult-isol*)

**moreover have**  $\dots \leq y$

**using**  $\langle z + y \cdot x \leq y \rangle$  **by** *auto*

**ultimately have**  $z \cdot x \wedge \text{Suc } n \leq y$  **by** *auto*}

**thus** ?case

**by** (*metis Suc*)

**qed**

**end**

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

**lemma** *power-is-relpow*: *rel-dioid.power* *X n* =  $X \wedge n$

**proof** (*induct n*)

**case** 0 **show** ?case

**by** (*metis rel-dioid.power-0 relpow.simps(1)*)

**case** *Suc* **thus** ?case

**by** (*metis rel-dioid.power-Suc2 relpow.simps(2)*)

**qed**

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

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

**lemma** *rel-star-contr*:  $X^* \circ Y = (\bigcup n. (\text{rel-dioid.power } X n) \circ Y)$   
**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**  $\text{Id} \cup x \circ x^* \subseteq x^*$

**by** (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)

**next**

**fix** *x y z* :: 'a *rel*

**assume**  $z \cup x \ O \ y \subseteq y$   
**thus**  $x^* \ O \ z \subseteq y$   
**by** (*simp only: rel-star-contr, metis (lifting) SUP-le-iff rel-diod.power-inductl*)  
**next**  
**fix**  $x \ y \ z :: 'a \ rel$   
**assume**  $z \cup y \ O \ x \subseteq y$   
**thus**  $z \ O \ x^* \subseteq y$   
**by** (*simp only: rel-star-contrl, metis (lifting) SUP-le-iff rel-diod.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**  
 $rel-R \ P \ Q = \bigcup \{X. rel-kat.Hoare \ P \ X \ Q\}$

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

## 0.9.2 Relational Store

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

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

**definition** *p2r* ::  $'a \ pred \Rightarrow 'a \ rel \ (\lceil \cdot \rceil)$  **where**  
 $\lceil P \rceil = \{(s, s) \mid s. P \ s\}$

**lemma** *p2r-simps*[*simp*]:  
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P \ s \longrightarrow Q \ s)$   
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P \ s = Q \ s)$   
 $(\lceil P \rceil ; \lceil Q \rceil) = \lceil \lambda s. P \ s \wedge Q \ s \rceil$   
 $(\lceil P \rceil \cup \lceil Q \rceil) = \lceil \lambda s. P \ s \vee Q \ s \rceil$   
 $rel-tests.t \ \lceil P \rceil = \lceil P \rceil$   
 $(- \ Id) \cup \lceil P \rceil = - \ \lceil \lambda s. \neg P \ s \rceil$   
 $Id \cap (- \ \lceil P \rceil) = \lceil \lambda s. \neg P \ s \rceil$   
**unfolding** *p2r-def* **by** *auto*

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

— Hoare Triples

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

**lemma** *rel-kat-H*:  $\{P\} \ X \ \{Q\} \longleftrightarrow (\forall s \ s'. P \ s \longrightarrow (s, s') \in X \longrightarrow Q \ s')$   
**by** (*simp add: rel-kat.Hoare-def, auto simp add: p2r-def*)

— Skip

**lemma** *sH-skip*[*simp*]:  $\{P\} \ skip \ \{Q\} \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$   
**unfolding** *rel-kat-H* **by** *simp*

**lemma** *H-skip*:  $\{P\} \ skip \ \{P\}$

**by** *simp*

— Abort

**lemma** *sH-abort*[*simp*]:  $\{P\} \text{ abort } \{Q\} \longleftrightarrow \text{True}$   
**unfolding** *rel-kat-H* **by** *simp*

**lemma** *H-abort*:  $\{P\} \text{ abort } \{Q\}$   
**by** *simp*

— Assignments

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

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

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

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

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

— Nondeterministic Assignments

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

**lemma** *sH-nondet-assign*[*simp*]:  $\{P\} (x ::= ?) \{Q\} \longleftrightarrow (\forall s. P \ s \longrightarrow (\forall k. Q \ (\chi j. (((\$) s)(x := k)) j)))$   
**unfolding** *rel-kat-H vec-upd-def nondet-assign-def* **by** (*auto simp: fun-upd-def*)

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

— Sequential Composition

**lemma** *H-seq*:  $\{P\} X \{R\} \Longrightarrow \{R\} Y \{Q\} \Longrightarrow \{P\} X; Y \{Q\}$   
**using** *rel-kat.H-seq* .

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

— Nondeterministic Choice

**lemma** *sH-choice*:  $\{P\} X \cup Y \{Q\} \longleftrightarrow (\{P\} X \{Q\} \wedge \{P\} Y \{Q\})$   
**unfolding** *rel-kat-H* **by** *auto*

**lemma** *H-choice*:  $\{P\} X \{Q\} \Longrightarrow \{P\} Y \{Q\} \Longrightarrow \{P\} X \cup Y \{Q\}$   
**using** *rel-kat.H-choice* .

— Conditional Statement

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



**lemma** *sH-cond[simp]*:

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

**lemma** *H-cond*:

$\{\lambda s. P s \wedge B s\} X \{Q\} \implies \{\lambda s. P s \wedge \neg B s\} Y \{Q\} \implies \{P\} (IF B THEN X ELSE Y) \{Q\}$   
**by** *simp*

— While Loop

**abbreviation** *while-inv-sugar* ::  $'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel}$

(*WHILE - INV - DO - [64,64,64] 63*)

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

**lemma** *sH-whileI*:  $\forall s. P s \longrightarrow I s \implies \forall s. I s \wedge \neg B s \longrightarrow Q s \implies \{\lambda s. I s \wedge B s\} X \{I\}$

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

**by** (*rule rel-kat.H-while-inv, auto simp: p2r-def rel-kat.Hoare-def, fastforce*)

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

**using** *rel-kat.H-while[of [P] [B] X]*

**unfolding** *rel-kat.kat-while-inv-def* **by** *auto*

— Finite Iteration

**abbreviation** *loopi-sugar* ::  $'a \text{ rel} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel} (LOOP - INV - [64,64] 63)$

**where** *LOOP X INV I*  $\equiv$  *rel-kat.kat-loop-inv X [I]*

**lemma** *H-loop*:  $\{P\} X \{P\} \implies \{P\} (LOOP X INV I) \{P\}$

**by** (*auto intro: rel-kat.H-loop*)

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

**using** *rel-kat.H-loop-inv[of [P] [I] X [Q]]* **by** *auto*

### 0.9.3 Verification of hybrid programs

— Verification by providing evolution

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

**where** *EVOL*  $\varphi G U = \{(s, s') \mid s s'. s' \in g\text{-orbit } (\lambda t. \varphi t s) G (U s)\}$

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

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

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

**unfolding** *rel-kat-H g-evol-def g-orbit-eq* **by** *auto*

**lemma** *H-g-evol*:

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

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

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

**by** (*simp add: assms*)

— Verification by providing solutions

**definition** *g-ode* ::  $(real \Rightarrow ('a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow ('a \Rightarrow real \text{ set}) \Rightarrow 'a \text{ set} \Rightarrow real \Rightarrow 'a \text{ rel} ((1x' = - \& - \text{ on } - \text{ @ } -))$

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

**lemma** *H-g-orbital*:

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

$\{P\} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \{Q\}$   
**unfolding** *rel-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

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

**context** *local-flow*

**begin**

**lemma** *sH-g-ode-subset*:

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

**proof**(*unfold sH-g-orbital, clarsimp, safe*)

**fix** *s t*

**assume** *hyps*:  $s \in S \ P \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (\varphi \ \tau \ s)$

**and** *main*:  $\forall s. P \ s \longrightarrow (\forall X \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s. \forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (X \ \tau)) \longrightarrow Q \ (X \ t))$

**hence**  $(\lambda t. \varphi \ t \ s) \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s$

**using** *in-ivp-sols assms* **by** *blast*

**thus**  $Q \ (\varphi \ t \ s)$

**using** *main hyps* **by** *fastforce*

**next**

**fix** *s X t*

**assume** *hyps*:  $P \ s \ X \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (X \ \tau)$

**and** *main*:  $\forall s \in S. P \ s \longrightarrow (\forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$

**hence** *obs*:  $s \in S$

**using** *ivp-sols-def[of  $\lambda t. f$ ] init-time* **by** *auto*

**hence**  $\forall \tau \in \text{down } (U \ s) \ t. X \ \tau = \varphi \ \tau \ s$

**using** *eq-solution hyps assms* **by** *blast*

**thus**  $Q \ (X \ t)$

**using** *hyps main obs* **by** *auto*

**qed**

**lemma** *H-g-ode-subset*:

**assumes**  $\bigwedge s. s \in S \implies 0 \in U \ s \wedge \text{is-interval } (U \ s) \wedge U \ s \subseteq T$   
**and**  $P = (\lambda s. s \in S \longrightarrow (\forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$

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

**using** *assms* **apply**(*subst sH-g-ode-subset[OF assms(1)]*)

**unfolding** *assms* **by** *auto*

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

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

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

**end**

— Verification with differential invariants

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

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

**lemma** *sH-g-orbital-guard*:

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

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

**using** *assms* **unfolding** *g-orbital-eq rel-kat-H ivp-sols-def g-ode-def* **by** *auto*

**lemma** *sH-g-orbital-inv*:

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

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

**using** *assms*(1) **apply**(*rule-tac*  $p' = \lceil I \rceil$  **in** *rel-kat.H-consl, simp*)

**using** *assms*(3) **apply**(*rule-tac*  $q' = \lceil I \rceil$  **in** *rel-kat.H-consr, simp*)

**using** *assms*(2) **by** *simp*

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

**unfolding** *diff-invariant-eq rel-kat-H g-orbital-eq g-ode-def* **by** *auto*

**lemma** *H-g-ode-inv*:  $\{I\} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \{I\} \implies \lceil P \rceil \leq \lceil I \rceil \implies$

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

**unfolding** *g-ode-inv-def* **apply**(*rule-tac*  $q' = \lceil \lambda s. \ I \ s \wedge \ G \ s \rceil$  **in** *rel-kat.H-consr, simp*)

**apply**(*subst sH-g-orbital-guard[symmetric], force*)

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

#### 0.9.4 Refinement Components

— Skip

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

**by** (*rule rel-rkat.R2, simp*)

— Abort

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

**by** (*rule rel-rkat.R2, simp*)

— Sequential Composition

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

**using** *rel-rkat.R-seq* **by** *blast*

**lemma** *R-seq-rule*:  $X \leq \text{rel-R } \lceil P \rceil \lceil R \rceil \implies Y \leq \text{rel-R } \lceil R \rceil \lceil Q \rceil \implies X ; Y \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$

**unfolding** *rel-rkat.spec-def* **by** (*rule H-seq*)

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

— Nondeterministic Choice

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

**using** *rel-rkat.R-choice[of \lceil P \rceil \lceil Q \rceil]* .

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

**using** *le-supI* .

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

**using** *Un-mono* .

— Assignment

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

**unfolding** *rel-rkat.spec-def* **by** (*rule H-assign, clarsimp simp: fun-upd-def*)

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

**lemma** *R-assignl*:  $P = (\lambda s. R\ (\chi\ j. (((\$)\ s)(x := e\ s)))\ j) \Longrightarrow (x ::= e) ; \text{rel-R}\ [\![R]\!] [\![Q]\!] \leq \text{rel-R}\ [\![P]\!] [\![Q]\!]$   
**apply** (rule-tac *R=R in R-seq-rule*)  
**by** (rule-tac *R-assign-rule, simp-all*)

**lemma** *R-assignr*:  $R = (\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := e\ s)))\ j) \Longrightarrow \text{rel-R}\ [\![P]\!] [\![R]\!]; (x ::= e) \leq \text{rel-R}\ [\![P]\!] [\![Q]\!]$   
**apply** (rule-tac *R=R in R-seq-rule, simp*)  
**by** (rule-tac *R-assign-rule, simp*)

**lemma**  $(x ::= e) ; \text{rel-R}\ [\![Q]\!] [\![Q]\!] \leq \text{rel-R}\ [(\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := e\ s)))\ j)] [\![Q]\!]$   
**by** (rule *R-assignl*) *simp*

**lemma**  $\text{rel-R}\ [\![Q]\!] [(\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := e\ s)))\ j)]; (x ::= e) \leq \text{rel-R}\ [\![Q]\!] [\![Q]\!]$   
**by** (rule *R-assignr*) *simp*

— Nondeterministic Assignment

**lemma** *R-nondet-assign*:  $(x ::= ?) \leq \text{rel-R}\ [\lambda s. \forall k. P\ (\chi\ j. (((\$)\ s)(x := k))\ j)] [\![P]\!]$   
**unfolding** *rel-rkat.spec-def* **by** (rule *H-nondet-assign*)

**lemma** *R-nondet-assign-rule*:  $(\forall s. P\ s \longrightarrow (\forall k. Q\ (\chi\ j. (((\$)\ s)(x := k))\ j))) \Longrightarrow (x ::= ?) \leq \text{rel-R}\ [\![P]\!] [\![Q]\!]$   
**unfolding** *sH-nondet-assign[symmetric]* **by** (rule *rel-rkat.R2*)

— Conditional Statement

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

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

— While Loop

**lemma** *R-while*:  $\text{WHILE } B \text{ INV } I \text{ DO } (\text{rel-R}\ [\lambda s. P\ s \wedge B\ s] [\![P]\!]) \leq \text{rel-R}\ [\![P]\!] [\lambda s. P\ s \wedge \neg B\ s]$   
**unfolding** *rel-kat.kat-while-inv-def* **using** *rel-rkat.R-while[of [B] [P]]* **by** *simp*

**lemma** *R-whileI*:

$X \leq \text{rel-R}\ [\![I]\!] [\![I]\!] \Longrightarrow [P] \leq [\lambda s. I\ s \wedge B\ s] \Longrightarrow [\lambda s. I\ s \wedge \neg B\ s] \leq [Q] \Longrightarrow \text{WHILE } B \text{ INV } I \text{ DO } X$   
 $\leq \text{rel-R}\ [\![P]\!] [\![Q]\!]$   
**by** (rule *rel-rkat.R2, rule rel-kat.H-while-inv, auto simp: rel-kat-H rel-rkat.spec-def*)

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

— Finite Iteration

**lemma** *R-loop:LOOP*  $\text{rel-R}\ [\![P]\!] [\![P]\!] \text{ INV } I \leq \text{rel-R}\ [\![P]\!] [\![P]\!]$   
**using** *rel-rkat.R-loop* .

**lemma** *R-loopI*:  $X \leq \text{rel-R}\ [\![I]\!] [\![I]\!] \Longrightarrow [P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow \text{LOOP } X \text{ INV } I \leq \text{rel-R}\ [\![P]\!] [\![Q]\!]$   
**unfolding** *rel-rkat.spec-def* **using** *H-loopI* **by** *blast*

**lemma** *R-loop-mono*:  $X \leq X' \Longrightarrow \text{LOOP } X \text{ INV } I \subseteq \text{LOOP } X' \text{ INV } I$   
**unfolding** *rel-kat.kat-loop-inv-def* **by** (*simp add: rel-ka.star-iso*)

— Evolution command (flow)

**lemma** *R-g-evol*:

**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $(EVOL \varphi G U) \leq rel-R [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow P (\varphi t s)] [P]$   
**unfolding** *rel-rkat.spec-def* **by** (*rule H-g-evol, simp*)

**lemma** *R-g-evol-rule*:

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

**lemma** *R-g-evoll*:

**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $P = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s)) \Longrightarrow (EVOL \varphi G U) ; rel-R [R] [Q] \leq rel-R [P] [Q]$   
**apply**(*rule-tac R=R in R-seq-rule*)  
**by** (*rule-tac R-g-evol-rule, simp-all*)

**lemma** *R-g-evolr*:

**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $R = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \Longrightarrow rel-R [P] [R] ; (EVOL \varphi G U) \leq rel-R [P] [Q]$   
**apply**(*rule-tac R=R in R-seq-rule, simp*)  
**by** (*rule-tac R-g-evol-rule, simp*)

**lemma**

**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $EVOL \varphi G U ; rel-R [Q] [Q] \leq rel-R [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)] [Q]$   
**by** (*rule R-g-evoll*) *simp*

**lemma**

**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $rel-R [Q] [\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)] ; EVOL \varphi G U \leq rel-R [Q] [Q]$   
**by** (*rule R-g-evolr*) *simp*

— Evolution command (ode)

**context** *local-flow*

**begin**

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

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

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

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

**lemma** *R-g-odel-subset*:

**assumes**  $\bigwedge s. s \in S \Longrightarrow 0 \in U s \wedge is-interval (U s) \wedge U s \subseteq T$   
**and**  $P = (\lambda s. \forall t \in U s. (\forall \tau \in down (U s) t. G (\varphi \tau s)) \longrightarrow R (\varphi t s))$

**shows**  $(x' = (\lambda t. f) \ \& \ G \text{ on } U \ S \ @ \ 0) ; \text{rel-}R \ [R] \ [Q] \leq \text{rel-}R \ [P] \ [Q]$   
**apply** (*rule-tac R=R in R-seq-rule, rule-tac R-g-ode-rule-subset*)  
**by** (*simp-all add: assms*)

**lemma** *R-g-oder-subset*:

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

**lemma** *R-g-ode*:

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

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

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

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

**lemma** *R-g-ode-ivl*:

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

**end**

— Evolution command (invariants)

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

### 0.9.5 Derivation of the rules of dL

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

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

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

**lemma** *diff-solve-rule1*:

**assumes** *local-flow*  $f \ \text{UNIV} \ \text{UNIV} \ \varphi$   
**and**  $\forall s. P \ s \longrightarrow (\forall t \geq 0. (\forall \tau \in \{0..t\}. G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$   
**shows**  $\{P\} \ (x' = f \ \& \ G) \ \{Q\}$   
**using** *assms* **by** (*subst local-flow.sH-g-ode-subset, auto*)

**lemma** *diff-solve-rule2*:

**fixes**  $c::a::\{\text{heine-borel}, \text{banach}\}$   
**assumes**  $\forall s. P \ s \longrightarrow (\forall t \geq 0. (\forall \tau \in \{0..t\}. G \ (s + \tau *_R c)) \longrightarrow Q \ (s + t *_R c))$

**shows**  $\{P\} (x' = (\lambda s. c) \ \& \ G) \{Q\}$   
**apply**(subst local-flow.sH-g-ode-subset[**where**  $T = \text{UNIV}$  **and**  $\varphi = (\lambda t \ x. x + t *_{\text{R}} c)$ ])  
**using** line-is-local-flow *assms* **by** *auto*

**lemma** *diff-weak-rule*:

**assumes**  $\lceil G \rceil \leq \lceil Q \rceil$   
**shows**  $\{P\} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \{Q\}$   
**using** *assms* **unfolding** g-orbital-eq rel-kat-H ivp-sols-def g-ode-def **by** *auto*

**lemma** *diff-cut-rule*:

**assumes**  $wp\text{-}C\text{:rel-kat.Hoare} \lceil P \rceil (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil C \rceil$   
**and**  $wp\text{-}Q\text{:rel-kat.Hoare} \lceil P \rceil (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil$   
**shows**  $\{P\} (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \{Q\}$   
**proof**(subst rel-kat-H, simp add: g-orbital-eq p2r-def g-ode-def, clarsimp)  
**fix**  $t::\text{real}$  **and**  $X::\text{real} \Rightarrow 'a$  **and**  $s$   
**assume**  $P \ s$  **and**  $t \in U \ s$   
**and**  $x\text{-ivp}: X \in \text{ivp-sols } f \ U \ S \ t_0 \ s$   
**and**  $\text{guard-}x: \forall x. x \in U \ s \wedge x \leq t \longrightarrow G \ (X \ x)$   
**have**  $\forall t \in (\text{down } (U \ s) \ t). X \ t \in g\text{-orbital } f \ G \ U \ S \ t_0 \ s$   
**using** g-orbitalI[OF  $x\text{-ivp}$ ]  $\text{guard-}x$  **by** *auto*  
**hence**  $\forall t \in (\text{down } (U \ s) \ t). C \ (X \ t)$   
**using**  $wp\text{-}C \ \langle P \ s \rangle$  **by** (subst (asm) rel-kat-H, auto simp: g-ode-def)  
**hence**  $X \ t \in g\text{-orbital } f \ (\lambda s. G \ s \wedge C \ s) \ U \ S \ t_0 \ s$   
**using**  $\text{guard-}x \ \langle t \in U \ s \rangle$  **by** (auto intro!: g-orbitalI  $x\text{-ivp}$ )  
**thus**  $Q \ (X \ t)$   
**using**  $\langle P \ s \rangle \ wp\text{-}Q$  **by** (subst (asm) rel-kat-H) (auto simp: g-ode-def)  
**qed**

**lemma** *diff-inv-rule*:

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

**end**

### 0.9.6 Examples

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

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

**begin**

#### Pendulum

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

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

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

— Verified with annotated dynamics

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

— Verified with differential invariants

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

— Verified with the flow

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

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

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

## Bouncing Ball

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

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

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

— Verified with differential invariants

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

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



**lemma** *fball-invariant*:

**fixes**  $g\ h :: \text{real}$

**defines** *dinv*:  $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$

**shows** *diff-invariant*  $I$   $(\lambda t. f\ g)$   $(\lambda s. \text{UNIV})$   $\text{UNIV } 0\ G$

**unfolding** *dinv* **apply**(*rule diff-invariant-rules*, *simp*)

**by**(*auto intro!*: *poly-derivatives*)

**lemma** *bouncing-ball-inv*:  $g < 0 \implies h \geq 0 \implies \text{rel-kat.Hoare}$

$[\lambda s. s\$1 = h \wedge s\$2 = 0]$

(*LOOP*

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

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

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

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

**apply**(*rule H-loopI*)

**apply**(*rule H-seq*[**where**  $R = \lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ])

**apply**(*rule H-g-ode-inv*)

**by** (*auto simp*: *bb-real-arith intro!*: *poly-derivatives diff-invariant-rules*)

— Verified with annotated dynamics

**lemma** [*bb-real-arith*]:

**assumes** *invar*:  $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$

**and** *pos*:  $g \cdot \tau^2 / 2 + v \cdot \tau + (x :: \text{real}) = 0$

**shows**  $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$

**and**  $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$

**proof**—

**from** *pos* **have**  $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$  **by** *auto*

**then** **have**  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$

**by** (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(1,3) *mult-zero-right*  
*monoid-mult-class.power2-eq-square semiring-class.distrib-left*)

**hence**  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot h = 0$

**using** *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)

**hence** *obs*:  $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$

**apply**(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(2, 3)  
*Groups.mult-ac*(2, 3) *monoid-mult-class.power2-eq-square nat-distrib*(2))

**thus**  $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$

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

**have**  $2 \cdot g \cdot h + (- ((g \cdot \tau) + v))^2 = 0$

**using** *obs* **by** (*metis* *Groups.add-ac*(2) *power2-minus*)

**thus**  $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$

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

**qed**

**lemma** [*bb-real-arith*]:

**assumes** *invar*:  $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$

**shows**  $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x :: \text{real})) =$

$2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$  (**is** *?lhs* = *?rhs*)

**proof**—

**have** *?lhs* =  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$

**by**(*auto simp*: *algebra-simps semiring-normalization-rules*(29))

**also** **have** ... =  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$  (**is** ... = *?middle*)

**by**(*subst invar*, *simp*)

**finally** **have** *?lhs* = *?middle*.

**moreover**

{**have** *?rhs* =  $g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$

**by** (*simp add*: *Groups.mult-ac*(2,3) *semiring-class.distrib-left*)

**also** **have** ... = *?middle*

**by** (*simp add*: *semiring-normalization-rules*(29))

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

**lemma** *bouncing-ball-dyn*:  $g < 0 \implies h \geq 0 \implies \text{rel-kat.Hoare}$   
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil$   
 $(\text{LOOP}$   
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$   
 $) \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
**apply**(*rule H-loopI*, *rule H-seq*[**where**  $R = \lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ])  
**by** (*auto simp: bb-real-arith*)

— Verified with the flow

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

**lemma** *bouncing-ball-flow*:  $g < 0 \implies h \geq 0 \implies \text{rel-kat.Hoare}$   
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil$   
 $(\text{LOOP}$   
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0));$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$   
 $) \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
**apply**(*rule H-loopI*)  
**apply**(*rule H-seq*[**where**  $R = \lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ])  
**apply**(*subst local-flow.sH-g-ode-subset[OF local-flow-ball]*, *simp*)  
**apply**(*force simp: bb-real-arith*)  
**by** (*rule H-cond*) (*auto simp: bb-real-arith*)

— Refined with annotated dynamics

**lemma** *R-bb-assign*:  $g < (0::\text{real}) \implies 0 \leq h \implies$   
 $2 ::= (\lambda s. - s\$2) \leq \text{rel-}R$   
 $\lceil \lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$   
 $\lceil \lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2 \rceil$   
**by** (*rule R-assign-rule, auto*)

**lemma** *R-bouncing-ball-dyn*:  
**assumes**  $g < 0$  **and**  $h \geq 0$   
**shows** *rel-R*  $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil \geq$   
 $(\text{LOOP}$   
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$   
**apply**(*rule order-trans*)  
**apply**(*rule R-loop-mono*) **defer**  
**apply**(*rule R-loopI*)  
**apply**(*rule R-seq*)  
**using** *assms* **apply**(*simp-all, force simp: bb-real-arith*)  
**apply**(*rule R-seq-mono*) **defer**  
**apply**(*rule order-trans*)  
**apply**(*rule R-cond-mono*) **defer defer**  
**apply**(*rule R-cond*) **defer**  
**using** *R-bb-assign* **apply** *force*

**apply**(*rule* *R-skip*, *clarsimp*)  
**by** (*rule* *R-g-evol-rule*, *force simp: bb-real-arith*)

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

## Thermostat

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

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

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

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

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

— Verified with the flow

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

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

**assume**  $a1: 0 < a$   
**have**  $f2: \bigwedge r \ ra. |(r::real) + -ra| = |ra + -r|$   
**by** (*metis abs-minus-commute minus-real-def*)  
**have**  $\bigwedge r \ ra \ rb. (r::real) * ra + - (r * rb) = r * (ra + -rb)$   
**by** (*metis minus-real-def right-diff-distrib*)  
**hence**  $|a * (s_1\$1 + -L) + - (a * (s_2\$1 + -L))| = a * |s_1\$1 + -s_2\$1|$   
**using**  $a1$  **by** (*simp add: abs-mult*)  
**thus**  $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$   
**using**  $f2$  *minus-real-def* **by** *presburger*

**qed**

**lemma** *local-lipschitz-therm-dyn*:

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

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

**by** (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-therm-dyn simp: forall-4 vec-eq-iff*)

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

**assumes**  $a > 0$  **and** *Thyps*:  $0 < Tmin \ Tmin \leq T \ T \leq Tmax$

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

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

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

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

**lemma** *thermostat-flow*:

assumes  $0 < a$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$   
 shows *rel-kat.Hoare* [*I Tmin Tmax*]  
 (*LOOP* (  
 — *control*  
 ( $2 ::= (\lambda s. 0)$ );  
 ( $3 ::= (\lambda s. s\$1)$ );  
 (*IF* ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1$ ) *THEN*  
 ( $4 ::= (\lambda s. 1)$ )  
*ELSE IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$ ) *THEN*  
 ( $4 ::= (\lambda s. 0)$ )  
*ELSE skip*);  
 — *dynamics*

```

(IF ( $\lambda s. s\$4 = 0$ ) THEN
  ( $x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ )
ELSE
  ( $x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ ))
) INV I Tmin Tmax
[I Tmin Tmax]
apply(rule H-loopI)
apply(rule-tac  $R = \lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0$  in H-seq, simp, simp)
apply(rule H-cond, simp-all add: H-g-ode-therm[OF assms(1,2)]) +
using therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin]
and therm-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

```

— Refined with the flow

**lemma** R-therm-dyn-down:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows  $rel\text{-}R \ [\lambda s. s\$4 = 0 \wedge I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$ 
  ( $x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ )
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

```

**lemma** R-therm-dyn-up:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows  $rel\text{-}R \ [\lambda s. s\$4 \neq 0 \wedge I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$ 
  ( $x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ )
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin] by auto

```

**lemma** R-therm-dyn:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows  $rel\text{-}R \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax] \geq$ 
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ )
  ELSE
    ( $x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0$ ))
apply(rule order-trans, rule R-cond-mono)
apply(rule R-therm-dyn-down[OF assms])
using R-therm-dyn-down[OF assms] R-therm-dyn-up[OF assms] by (auto intro!: R-cond)

```

**lemma** R-therm-assign1:  $rel\text{-}R \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \geq (2 ::= (\lambda s. 0))$   
**by** (auto simp: R-assign-rule)

**lemma** R-therm-assign2:

```

 $rel\text{-}R \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq (3 ::= (\lambda s. s\$1))$ 
by (auto simp: R-assign-rule)

```

**lemma** R-therm-ctrl:

```

 $rel\text{-}R \ [I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$ 
  ( $2 ::= (\lambda s. 0)$ );
  ( $3 ::= (\lambda s. s\$1)$ );
  (IF ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1$ ) THEN
    ( $4 ::= (\lambda s. 1)$ )
  ELSE IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$ ) THEN
    ( $4 ::= (\lambda s. 0)$ )
  ELSE skip)
apply(rule R-seq-rule) +
apply(rule R-therm-assign1)
apply(rule R-therm-assign2)
apply(rule order-trans)

```

```

apply(rule R-cond-mono)
apply(rule R-assign-rule) defer
apply(rule R-cond-mono)
apply(rule R-assign-rule) defer
apply(rule R-skip) defer
apply(rule order-trans)
apply(rule R-cond-mono)
apply force
by (rule R-cond)+ auto

```

**lemma** *R-therm-loop*:  $\text{rel-}R \ [I \ Tmin \ Tmax] \ [I \ Tmin \ Tmax] \geq$   
 (*LOOP* (*rel-R*  $[I \ Tmin \ Tmax] \ [\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1]$ ;  
*rel-R*  $[\lambda s. I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax]$   
*INV*  $I \ Tmin \ Tmax$ )  
**by** (*intro R-loopI R-seq, simp-all*)

**lemma** *R-thermostat-flow*:  
**assumes**  $a > 0$  **and**  $0 \leq \tau$  **and**  $0 < Tmin$  **and**  $Tmax < L$   
**shows**  $\text{rel-}R \ [I \ Tmin \ Tmax] \ [I \ Tmin \ Tmax] \geq$   
 (*LOOP* (  
 — control  
 ( $2 ::= (\lambda s. 0)$ );( $3 ::= (\lambda s. s\$1)$ );  
 (*IF*  $(\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1)$  *THEN*  
 ( $4 ::= (\lambda s. 1)$ )  
*ELSE IF*  $(\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1)$  *THEN*  
 ( $4 ::= (\lambda s. 0)$ )  
*ELSE skip*);  
 — dynamics  
 (*IF*  $(\lambda s. s\$4 = 0)$  *THEN*  
 ( $x' = (\lambda t. f \ a \ 0) \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on} \ (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0$ )  
*ELSE*  
 ( $x' = (\lambda t. f \ a \ L) \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on} \ (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0$ ))  
 ) *INV*  $I \ Tmin \ Tmax$ )  
**by** (*intro order-trans[OF - R-therm-loop] R-loop-mono*  
*R-seq-mono R-therm-ctrl R-therm-dyn[OF assms]*)

**no-notation** *therm-vec-field* ( $f$ )  
**and** *therm-flow* ( $\varphi$ )  
**and** *therm-guard* ( $G$ )  
**and** *therm-loop-inv* ( $I$ )

## Water tank

— Variation of Hespanha and [?]

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

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

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

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

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

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

— Verified with the flow

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

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

**lemma** *tank-arith*:

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

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

**lemma** *tank-flow*:

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

— Verified with differential invariants

**lemma** *tank-diff-inv*:

$0 \leq \tau \implies \text{diff-invariant } (dI \text{ hmin hmax } k) (\lambda t. f \ k) (\lambda s. \{0..\tau\}) \text{ UNIV } 0 \text{ Guard}$   
**apply**(*intro diff-invariant-conj-rule*)  
**apply**(*force intro*!: *poly-derivatives diff-invariant-rules*)  
**apply**(*rule-tac*  $\nu'=\lambda t. 0$  **and**  $\mu'=\lambda t. 1$  **in** *diff-invariant-leq-rule*, *simp-all*, *presburger*)  
**apply**(*rule-tac*  $\nu'=\lambda t. 0$  **and**  $\mu'=\lambda t. 0$  **in** *diff-invariant-leq-rule*, *simp-all*)  
**apply**(*force intro*!: *poly-derivatives*)  
**by** (*auto intro*!: *poly-derivatives diff-invariant-rules*)

**lemma** *tank-inv-arith1*:

**assumes**  $0 \leq (\tau::\text{real})$  **and**  $c_o < c_i$  **and**  $b: \text{hmin} \leq y_0$  **and**  $g: \tau \leq (\text{hmax} - y_0) / (c_i - c_o)$   
**shows**  $\text{hmin} \leq (c_i - c_o) \cdot \tau + y_0$  **and**  $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$   
**proof**—  
**have**  $(c_i - c_o) \cdot \tau \leq (\text{hmax} - y_0)$   
**using**  $g$  *assms(2,3)* **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)  
**thus**  $(c_i - c_o) \cdot \tau + y_0 \leq \text{hmax}$

by *auto*  
 show  $hmin \leq (c_i - c_o) \cdot \tau + y_0$   
 using  $b$  *assms*(1,2) by (*metis add.commute add-increasing2 diff-ge-0-iff-ge less-eq-real-def mult-nonneg-nonneg*)  
 qed

lemma *tank-inv-arith2*:

assumes  $0 \leq (\tau :: \text{real})$  and  $0 < c_o$  and  $b: y_0 \leq hmax$  and  $g: \tau \leq -((hmin - y_0) / c_o)$   
 shows  $hmin \leq y_0 - c_o \cdot \tau$  and  $y_0 - c_o \cdot \tau \leq hmax$

proof—

have  $\tau \cdot c_o \leq y_0 - hmin$   
 using  $g \langle 0 < c_o \rangle$  *pos-le-minus-divide-eq* by *fastforce*  
 thus  $hmin \leq y_0 - c_o \cdot \tau$   
 by (*auto simp: mult.commute*)  
 show  $y_0 - c_o \cdot \tau \leq hmax$   
 using  $b$  *assms*(1,2) by (*smt mult-nonneg-nonneg*)  
 qed

lemma *tank-inv*:

assumes  $0 \leq \tau$  and  $0 < c_o$  and  $c_o < c_i$

shows *rel-kat.Hoare*  $\lceil I \ hmin \ hmax \rceil$

(*LOOP*

— control

(( $2 ::= (\lambda s. 0)$ ));( $3 ::= (\lambda s. s\$1)$ );  
 (*IF* ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ ) *THEN* ( $4 ::= (\lambda s. 1)$ ) *ELSE*  
 (*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) *THEN* ( $4 ::= (\lambda s. 0)$ ) *ELSE skip*));

— dynamics

(*IF* ( $\lambda s. s\$4 = 0$ ) *THEN*

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

*ELSE*

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

*INV I hmin hmax*)  $\lceil I \ hmin \ hmax \rceil$

apply(*rule H-loopI*)

apply(*rule-tac R*= $\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0 \wedge s\$3 = s\$1$  *in H-seq*)

apply(*rule-tac R*= $\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0 \wedge s\$3 = s\$1$  *in H-seq*)

apply(*rule-tac R*= $\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0$  *in H-seq, simp, simp*)

apply(*rule H-cond, simp*)

apply(*rule H-cond, simp, simp*)

apply(*rule H-cond*)

apply(*rule H-g-ode-inv*)

using *assms tank-inv-arith1* apply(*force simp: tank-diff-inv, simp, clarsimp*)

apply(*rule H-g-ode-inv*)

using *assms tank-diff-inv* [*of - -c\_o hmin hmax*] *tank-inv-arith2* by *auto*

— Refined with differential invariants

lemma *R-tank-inv*:

assumes  $0 \leq \tau$  and  $0 < c_o$  and  $c_o < c_i$

shows *rel-R*  $\lceil I \ hmin \ hmax \rceil \ \lceil I \ hmin \ hmax \rceil \geq$

(*LOOP*

— control

(( $2 ::= (\lambda s. 0)$ ));( $3 ::= (\lambda s. s\$1)$ );  
 (*IF* ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ ) *THEN* ( $4 ::= (\lambda s. 1)$ ) *ELSE*  
 (*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) *THEN* ( $4 ::= (\lambda s. 0)$ ) *ELSE skip*));

— dynamics

(*IF* ( $\lambda s. s\$4 = 0$ ) *THEN*

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

*ELSE*

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

*INV I hmin hmax*) (*is LOOP* (*?ctrl;?dyn*) *INV -*  $\leq$  *?ref*)



**proof**—

— First we refine the control.

```

let ?Ictrl =  $\lambda s. I \text{ hmin } hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ 
and ?cond =  $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ 
have ifbranch1:  $4 ::= (\lambda s. 1) \leq rel\text{-}R \ [\lambda s. ?cond \ s \wedge ?Ictrl \ s] \ [\ ?Ictrl] \ (\text{is} \ - \leq ?branch1)$ 
  by (rule R-assign-rule, simp)
have ifbranch2:  $(IF \ (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \ THEN \ (4 ::= (\lambda s. 0)) \ ELSE \ skip) \leq$ 
   $rel\text{-}R \ [\lambda s. \neg ?cond \ s \wedge ?Ictrl \ s] \ [\ ?Ictrl] \ (\text{is} \ - \leq ?branch2)$ 
  apply(rule order-trans, rule R-cond-mono) defer defer
  by (rule R-cond) (auto intro!: R-assign-rule R-skip)
have ifthenelse:  $(IF \ ?cond \ THEN \ ?branch1 \ ELSE \ ?branch2) \leq rel\text{-}R \ [\ ?Ictrl] \ [\ ?Ictrl] \ (\text{is} \ ?ifthenelse \leq$ 
   $-)$ 
  by (rule R-cond)
have  $(IF \ ?cond \ THEN \ (4 ::= (\lambda s. 1)) \ ELSE \ (IF \ (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \ THEN \ (4 ::= (\lambda s. 0)) \ ELSE \ skip)) \leq$ 
   $rel\text{-}R \ [\ ?Ictrl] \ [\ ?Ictrl]$ 
  apply(rule-tac  $y=?ifthenelse$  in order-trans, rule R-cond-mono)
  using ifbranch1 ifbranch2 ifthenelse by auto
hence ctrl:  $?ctrl \leq rel\text{-}R \ [I \text{ hmin } hmax] \ [\ ?Ictrl]$ 
  apply(rule-tac  $R=?Ictrl$  in R-seq-rule)
  apply(rule-tac  $R=\lambda s. I \text{ hmin } hmax \ s \wedge s\$2 = 0$  in R-seq-rule)
  by (auto intro!: R-assign-rule)
— Then we refine the dynamics.
have dynup:  $(x' = (\lambda t. f \ (c_i - c_o)) \ \& \ G \ hmax \ (c_i - c_o) \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax$ 
   $(c_i - c_o))) \leq$ 
   $rel\text{-}R \ [\lambda s. s\$4 = 0 \wedge ?Ictrl \ s] \ [I \text{ hmin } hmax]$ 
  apply(rule R-g-ode-inv[OF tank-diff-inv[OF assms(1)]])
  using assms by (auto simp: tank-inv-arith1)
have dyndown:  $(x' = (\lambda t. f \ (-c_o)) \ \& \ G \ hmin \ (-c_o) \ on \ (\lambda s. \{0..\tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax$ 
   $(-c_o))) \leq$ 
   $rel\text{-}R \ [\lambda s. s\$4 \neq 0 \wedge ?Ictrl \ s] \ [I \text{ hmin } hmax]$ 
  apply(rule R-g-ode-inv)
  using tank-diff-inv[OF assms(1), of  $-c_o$ ] assms
  by (auto simp: tank-inv-arith2)
have dyn:  $?dyn \leq rel\text{-}R \ [\ ?Ictrl] \ [I \text{ hmin } hmax]$ 
  apply(rule order-trans, rule R-cond-mono)
  using dynup dyndown by (auto intro!: R-cond)
— Finally we put everything together.
have pre-inv:  $[I \text{ hmin } hmax] \leq [I \text{ hmin } hmax]$ 
  by simp
have inv-pos:  $[I \text{ hmin } hmax] \leq [\lambda s. hmin \leq s\$1 \wedge s\$1 \leq hmax]$ 
  by simp
have inv-inv:  $rel\text{-}R \ [I \text{ hmin } hmax] \ [\ ?Ictrl]; (rel\text{-}R \ [\ ?Ictrl] \ [I \text{ hmin } hmax]) \leq rel\text{-}R \ [I \text{ hmin } hmax] \ [I$ 
   $hmin \ hmax]$ 
  by (rule R-seq)
have loopref:  $LOOP \ rel\text{-}R \ [I \text{ hmin } hmax] \ [\ ?Ictrl]; (rel\text{-}R \ [\ ?Ictrl] \ [I \text{ hmin } hmax]) \ INV \ I \text{ hmin } hmax$ 
   $\leq ?ref$ 
  apply(rule R-loopI)
  using pre-inv inv-inv inv-pos by auto
have obs:  $?ctrl; ?dyn \leq rel\text{-}R \ [I \text{ hmin } hmax] \ [\ ?Ictrl]; (rel\text{-}R \ [\ ?Ictrl] \ [I \text{ hmin } hmax])$ 
  apply(rule R-seq-mono)
  using ctrl dyn by auto
show  $LOOP \ (?ctrl; ?dyn) \ INV \ I \text{ hmin } hmax \leq ?ref$ 
  by (rule order-trans[OF - loopref], rule R-loop-mono[OF obs])
qed

```

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

**and** tank-flow ( $\varphi$ )

**and** tank-guard ( $G$ )

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

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

**end**

## 0.10 Verification and refinement of HS in the relational KAT

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

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

**imports**

*HS-VC-KAT*

*../HS-ODEs*

*Transformer-Semantics.Kleisli-Quantale*

**begin**

### 0.10.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

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

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

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

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

**and** *Archimedean-Field.floor-ceiling-class.floor* ( $(\lfloor - \rfloor)$ )

**and** *tau* ( $\tau$ )

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

**and** *proto-near-quantale-class.bres* (**infixr**  $\rightarrow$  60)

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

**apply** (*subgoal-tac* *Rep-nd-fun*  $f = \text{Rep-nd-fun } g$ )

**using** *Rep-nd-fun-inject*

**apply** *blast*

**by** (*rule ext*, *simp*)

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

**by** (*auto simp: nd-fun-ext*)

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

**begin**

**definition**  $0 = \zeta^\bullet$

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

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

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

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

**lemma** *nd-fun-plus-assoc*[*nd-fun-aka*]:  $x + y + z = x + (y + z)$

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

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

**lemma** *nd-fun-plus-zero*[*nd-fun-aka*]:  $0 + x = x$   
**and** *nd-fun-mult-zero*[*nd-fun-aka*]:  $0 \cdot x = 0$   
**and** *nd-fun-mult-zero*[*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*]:  $1 + x \cdot x^* \leq x^*$   
**and** *nd-star-unfoldl*[*nd-fun-aka*]:  $z + x \cdot y \leq y \implies x^* \cdot z \leq y$   
**and** *nd-star-unfoldr*[*nd-fun-aka*]:  $z + y \cdot x \leq y \implies z \cdot x^* \leq y$  **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**  $n\ f = (\lambda x. \text{if } ((f \bullet) x = \{\}) \text{ then } \{x\} \text{ else } \{\})^\bullet$

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

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

**end**

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

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

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

**end**

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

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

**lemma**  $p2ndf\text{-simps}[simp]:$

$\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P\ s \longrightarrow Q\ s)$   
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P\ s = Q\ s)$   
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P\ s \wedge Q\ s \rceil$   
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P\ s \vee Q\ s \rceil$   
 $\text{tt } \lceil P \rceil = \lceil P \rceil$   
 $n\ \lceil P \rceil = \lceil \lambda s. \neg P\ s \rceil$

**unfolding**  $p2ndf\text{-def one-nd-fun-def less-eq-nd-fun-def times-nd-fun-def plus-nd-fun-def}$   
**by**  $(\text{auto simp: nd-fun-eq-iff kcomp-def le-fun-def n-op-nd-fun-def})$

— Meaning of the state-transformer Hoare triple

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

— Hoare triple for skip and a simp-rule

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

**lemma**  $H\text{-skip}: \text{Hoare } \lceil P \rceil\ \text{skip}\ \lceil P \rceil$   
**using**  $H\text{-skip}$  **by**  $\text{blast}$

**lemma**  $sH\text{-skip}[simp]: \text{Hoare } \lceil P \rceil\ \text{skip}\ \lceil Q \rceil \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$   
**unfolding**  $ndfun\text{-kat-H}$  **by**  $(\text{simp add: one-nd-fun-def})$

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

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

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

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

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

— Next, the Hoare rule of the composition

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

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

**lemma**  $sH\text{-seq}: \text{Hoare } \lceil P \rceil\ (X\ ;\ Y)\ \lceil Q \rceil = \text{Hoare } \lceil P \rceil\ (X)\ \lceil \lambda s. \forall s'. s' \in (Y_\bullet)\ s \longrightarrow Q\ s' \rceil$   
**unfolding**  $ndfun\text{-kat-H}$  **by**  $(\text{auto simp: times-nd-fun-def kcomp-def})$

— Rewriting the Hoare rule for the conditional statement

**abbreviation**  $\text{cond-sugar} :: 'a \text{ pred} \Rightarrow 'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun} \Rightarrow 'a \text{ nd-fun } (\text{IF} - \text{THEN} - \text{ELSE} - [64, 64])$

63)

**where**  $IF\ B\ THEN\ X\ ELSE\ Y \equiv kat\text{-}cond\ [B]\ X\ Y$ 

**lemma**  $H\text{-}cond$ :  $Hoare\ [\lambda s. P\ s \wedge B\ s]\ X\ [Q] \implies Hoare\ [\lambda s. P\ s \wedge \neg B\ s]\ Y\ [Q] \implies$   
 $Hoare\ [P]\ (IF\ B\ THEN\ X\ ELSE\ Y)\ [Q]$   
**by** (rule  $H\text{-}cond$ ,  $simp\text{-}all$ )

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

— Rewriting the Hoare rule for the while loop

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

**where**  $WHILE\ B\ INV\ I\ DO\ X \equiv kat\text{-}while\text{-}inv\ [B]\ [I]\ X$ 

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

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

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

**lemma**  $H\text{-}loop$ :  $Hoare\ [P]\ X\ [P] \implies Hoare\ [P]\ (LOOP\ X\ INV\ I)\ [P]$   
**by** (auto  $intro$ :  $H\text{-}loop$ )

**lemma**  $H\text{-}loopI$ :  $Hoare\ [I]\ X\ [I] \implies [P] \leq [I] \implies [I] \leq [Q] \implies Hoare\ [P]\ (LOOP\ X\ INV\ I)\ [Q]$   
**using**  $H\text{-}loop\text{-}inv[of\ [P]\ [I]\ X\ [Q]]$  **by** auto

## 0.10.2 Verification of hybrid programs

— Verification by providing evolution

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

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

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

— Verification by providing solutions

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

**lemma**  $H\text{-}g\text{-}orbital$ :  
 $P = (\lambda s. (\forall X \in ivp\text{-}sols\ f\ U\ S\ t_0\ s. \forall t \in U\ s. (\forall \tau \in down\ (U\ s)\ t. G\ (X\ \tau)) \longrightarrow Q\ (X\ t))) \implies$   
 $Hoare\ [P]\ (x' = f \& G\ on\ U\ S\ @\ t_0)\ [Q]$

**unfolding** *ndfun-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

**lemma** *sH-g-orbital*: *Hoare*  $\lceil P \rceil (x' = f \ \& \ G \text{ on } U \ S \ @ \ t_0) \lceil Q \rceil =$   
 $(\forall s. P \ s \longrightarrow (\forall X \in \text{ivp-sols } f \ U \ S \ t_0 \ s. \forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G \ (X \ \tau)) \longrightarrow Q \ (X \ t)))$   
**unfolding** *g-orbital-eq g-ode-def ndfun-kat-H* **by** *auto*

**context** *local-flow*  
**begin**

**lemma** *sH-g-ode-subset*:

**assumes**  $\bigwedge s. s \in S \implies 0 \in U \ s \wedge \text{is-interval } (U \ s) \wedge U \ s \subseteq T$   
**shows** *Hoare*  $\lceil P \rceil (x' = (\lambda t. f) \ \& \ G \text{ on } U \ S \ @ \ 0) \lceil Q \rceil =$   
 $(\forall s \in S. P \ s \longrightarrow (\forall t \in U \ s. (\forall \tau \in \text{down } (U \ s) \ t. G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$

**proof**(*unfold sH-g-orbital, clarsimp, safe*)

**fix** *s t*

**assume** *hyps*:  $s \in S \ P \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (\varphi \ \tau \ s)$

**and** *main*:  $\forall s. P \ s \longrightarrow (\forall X \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s. \forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (X \ \tau)) \longrightarrow Q \ (X \ t))$

**hence**  $(\lambda t. \varphi \ t \ s) \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s$

**using** *in-ivp-sols assms* **by** *blast*

**thus**  $Q \ (\varphi \ t \ s)$

**using** *main hyps* **by** *fastforce*

**next**

**fix** *s X t*

**assume** *hyps*:  $P \ s \ X \in \text{Sols } (\lambda t. f) \ U \ S \ 0 \ s \ t \in U \ s \ \forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (X \ \tau)$

**and** *main*:  $\forall s \in S. P \ s \longrightarrow (\forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$

**hence** *obs*:  $s \in S$

**using** *ivp-sols-def[of \lambda t. f] init-time* **by** *auto*

**hence**  $\forall \tau \in \text{down } (U \ s) \ t. X \ \tau = \varphi \ \tau \ s$

**using** *eq-solution hyps assms* **by** *blast*

**thus**  $Q \ (X \ t)$

**using** *hyps main obs* **by** *auto*

**qed**

**lemma** *H-g-ode-subset*:

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

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

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

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

**end**

— Verification with differential invariants

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

**lemma** *sH-g-orbital-guard*:

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

**shows**  $\text{Hoare } [P] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [Q] = \text{Hoare } [P] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [R]$

**using** *assms unfolding g-orbital-eq ndfun-kat-H inv-sols-def g-ode-def* **by** *auto*

**lemma** *sH-g-orbital-inv*:

**assumes**  $[P] \leq [I]$  **and**  $\text{Hoare } [I] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [I]$  **and**  $[I] \leq [Q]$

**shows**  $\text{Hoare } [P] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [Q]$

**using** *assms(1) apply(rule-tac p'=[I] in H-consl, simp)*

**using** *assms(3) apply(rule-tac q'=[I] in H-consr, simp)*

**using** *assms(2) by simp*

**lemma** *sH-diff-inv[simp]*:  $\text{Hoare } [I] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [I] = \text{diff-invariant } I\ f\ T\ S\ t_0\ G$

**unfolding** *diff-invariant-eq ndfun-kat-H g-orbital-eq g-ode-def* **by** *auto*

**lemma** *H-g-ode-inv*:  $\text{Hoare } [I] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0) [I] \implies [P] \leq [I] \implies$

$[\lambda s. I\ s \wedge G\ s] \leq [Q] \implies \text{Hoare } [P] (x' = f \ \& \ G \text{ on } T\ S \ @ \ t_0\ \text{DINV } I) [Q]$

**unfolding** *g-ode-inv-def* **apply**(*rule-tac q'=[λs. I s ∧ G s] in H-consr, simp*)

**apply**(*subst sH-g-orbital-guard[symmetric], force*)

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

### 0.10.3 Refinement Components

— Skip

**lemma** *R-skip*:  $(\forall s. P\ s \longrightarrow Q\ s) \implies 1 \leq \text{Ref } [P] [Q]$

**by** (*auto simp: spec-def ndfun-kat-H one-nd-fun-def*)

— Composition

**lemma** *R-seq*:  $(\text{Ref } [P] [R]) ; (\text{Ref } [R] [Q]) \leq \text{Ref } [P] [Q]$

**using** *R-seq* **by** *blast*

**lemma** *R-seq-rule*:  $X \leq \text{Ref } [P] [R] \implies Y \leq \text{Ref } [R] [Q] \implies X ; Y \leq \text{Ref } [P] [Q]$

**unfolding** *spec-def* **by** (*rule H-seq*)

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

— Assignment

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

**unfolding** *spec-def* **by** (*rule H-assign, clarsimp simp: fun-eq-iff fun-upd-def*)

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

**unfolding** *sH-assign[symmetric] spec-def* .

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

**apply**(*rule-tac R=R in R-seq-rule*)

**by** (*rule-tac R-assign-rule, simp-all*)

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

**apply**(*rule-tac R=R in R-seq-rule, simp*)

**by** (*rule-tac R-assign-rule, simp*)

**lemma**  $(x ::= e) ; \text{Ref } [Q] [Q] \leq \text{Ref } [(\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := e\ s))\ j))] [Q]$

**by** (*rule R-assignl*) *simp*

**lemma**  $\text{Ref } [Q] [(\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := e\ s))\ j))] ; (x ::= e) \leq \text{Ref } [Q] [Q]$

**by** (*rule R-assignr*) *simp*

— Conditional

**lemma** *R-cond*:  $(IF\ B\ THEN\ Ref\ [\lambda s. B\ s \wedge P\ s]\ [Q]\ ELSE\ Ref\ [\lambda s. \neg B\ s \wedge P\ s]\ [Q]) \leq Ref\ [P]\ [Q]$   
**using** *R-cond*[of  $[B]\ [P]\ [Q]$ ] **by** *simp*

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

— While loop

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

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

— Finite loop

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

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

— Evolution command (flow)

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

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

**lemma** *R-g-evoll*:  
**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $P = (\lambda s. \forall t \in U\ s. (\forall \tau \in down\ (U\ s)\ t. G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \implies (EVOL\ \varphi\ G\ U) ; Ref\ [R]\ [Q] \leq Ref\ [P]\ [Q]$   
**apply**(*rule-tac R=R in R-seq-rule*)  
**by** (*rule-tac R-g-evol-rule, simp-all*)

**lemma** *R-g-evolr*:  
**fixes**  $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$   
**shows**  $R = (\lambda s. \forall t \in U\ s. (\forall \tau \in down\ (U\ s)\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \implies Ref\ [P]\ [R]; (EVOL\ \varphi\ G\ U) \leq Ref\ [P]\ [Q]$   
**apply**(*rule-tac R=R in R-seq-rule, simp*)  
**by** (*rule-tac R-g-evol-rule, simp*)

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

**lemma**



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

— Evolution command (ode)

**context** *local-flow*

**begin**

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

**assumes**  $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$   
**shows**  $(x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) \leq \text{Ref } \lceil \lambda s. s \in S \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow P (\varphi t s)) \rceil \lceil P \rceil$   
**unfolding** *spec-def* **by** (rule *H-g-ode-subset[OF assms]*, *auto*)

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

**assumes**  $\bigwedge s. s \in S \implies 0 \in U s \wedge \text{is-interval } (U s) \wedge U s \subseteq T$   
**shows**  $(\forall s \in S. P s \longrightarrow (\forall t \in U s. (\forall \tau \in \text{down } (U s) t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s))) \implies (x' = (\lambda t. f) \ \& \ G \text{ on } U S @ 0) \leq \text{Ref } \lceil P \rceil \lceil Q \rceil$   
**unfolding** *spec-def* **by** (*subst sH-g-ode-subset[OF assms]*, *auto*)

**lemma** *R-g-odel-subset*:

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

**lemma** *R-g-oder-subset*:

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

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

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

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

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

**lemma** *R-g-ode-ivl*:

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

**end**

— Evolution command (invariants)

**lemma** *R-g-ode-inv*: *diff-invariant*  $I f T S t_0 G \implies [P] \leq [I] \implies [\lambda s. I s \wedge G s] \leq [Q] \implies$   
 $(x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I) \leq \text{Ref } [P] [Q]$   
**unfolding** *spec-def* **by** (*auto simp: H-g-ode-inv*)

#### 0.10.4 Derivation of the rules of dL

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

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

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

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

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

**lemma** *diff-weak-rule*:  
**assumes**  $[G] \leq [Q]$   
**shows** *Hoare*  $[P] (x' = f \ \& \ G \text{ on } T S @ t_0) [Q]$   
**using** *assms* **unfolding** *g-orbital-eq ndfun-kat-H ivp-sols-def g-ode-def* **by** *auto*

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

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

**end**

### 0.10.5 Examples

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

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

**begin**

#### Pendulum

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

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

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

— Verified with annotated dynamics

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

— Verified with differential invariants

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

— Verified with the flow

**lemma** *local-flow-pend*:  $local-flow \ f \ UNIV \ UNIV \ \varphi$   
**apply**(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff, clarsimp*)  
**apply**(*rule-tac x=1 in exI, clarsimp, rule-tac x=1 in exI*)  
**apply**(*simp add: dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)  
**by** *(auto simp: forall-2 intro!: poly-derivatives)*

**lemma** *pendulum-flow*:  $Hoare \ [\lambda s. r^2 = (s\$1)^2 + (s\$2)^2] \ (x'=f \ \& \ G) \ [\lambda s. r^2 = (s\$1)^2 + (s\$2)^2]$   
**by** *(subst local-flow.sH-g-ode-subset[OF local-flow-pend], simp-all)*

**no-notation** *fpend* ( $f$ )  
**and** *pend-flow* ( $\varphi$ )

#### Bouncing Ball

A ball is dropped from rest at an initial height  $h$ . The motion is described with the free-fall equations  $x' t = v t$  and  $v' t = g$  where  $g$  is the constant acceleration due to gravity. The bounce is modelled with a variable assignment that flips the velocity, thus it is a completely elastic collision with the ground. We use  $s\$1$  to ball's height and  $s\$2$  for its velocity. We prove that the ball remains above ground and below its initial resting position.

**abbreviation** *fball* ::  $real \Rightarrow real^2 \Rightarrow real^2 (f)$   
**where**  $f g s \equiv (\chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } g)$

**abbreviation** *ball-flow* ::  $real \Rightarrow real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$   
**where**  $\varphi g \tau s \equiv (\chi i. \text{if } i=1 \text{ then } g \cdot \tau^2/2 + s\$2 \cdot \tau + s\$1 \text{ else } g \cdot \tau + s\$2)$

— Verified with differential invariants

**named-theorems** *bb-real-arith* real arithmetic properties for the bouncing ball.

**lemma** [*bb-real-arith*]:

**assumes**  $0 > g$  **and** *inv*:  $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$   
**shows**  $(x::\text{real}) \leq h$

**proof**—

**have**  $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$   
**using** *inv* **and**  $\langle 0 > g \rangle$  **by** *auto*  
**hence** *obs*:  $v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$   
**using** *left-diff-distrib* *mult.commute* **by** (*metis zero-le-square*)  
**hence**  $(v \cdot v)/(2 \cdot g) = (x - h)$   
**by** *auto*  
**also from** *obs* **have**  $(v \cdot v)/(2 \cdot g) \leq 0$   
**using** *divide-nonneg-neg* **by** *fastforce*  
**ultimately have**  $h - x \geq 0$   
**by** *linarith*  
**thus** *?thesis* **by** *auto*

**qed**

**lemma** *fball-invariant*:

**fixes**  $g \ h :: \text{real}$   
**defines** *dinv*:  $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$   
**shows** *diff-invariant*  $I$   $(\lambda t. f \ g)$   $(\lambda s. \text{UNIV})$   $\text{UNIV}$   $0 \ G$   
**unfolding** *dinv* **apply**(*rule diff-invariant-rules*, *simp*)  
**by**(*auto intro!*: *poly-derivatives*)

**lemma** *bouncing-ball-inv*:  $g < 0 \implies h \geq 0 \implies \text{Hoare}$

$\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil$   
 $(\text{LOOP}$   
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0) \ \text{DINV} \ (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2 = 0));$   
 $(\text{IF} \ (\lambda s. s\$1 = 0) \ \text{THEN} \ (2 ::= (\lambda s. - s\$2)) \ \text{ELSE} \ \text{skip}))$   
 $\text{INV} \ (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$   
 $\rceil \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$   
**apply**(*rule H-loopI*)  
**apply**(*rule H-seq*[**where**  $R = \lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ])  
**apply**(*rule H-g-ode-inv*)  
**by** (*auto simp*: *bb-real-arith intro!*: *poly-derivatives diff-invariant-rules*)

— Verified with annotated dynamics

**lemma** [*bb-real-arith*]:

**assumes** *invar*:  $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$   
**and** *pos*:  $g \cdot \tau^2 / 2 + v \cdot \tau + (x::\text{real}) = 0$   
**shows**  $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$   
**and**  $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$

**proof**—

**from** *pos* **have**  $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$  **by** *auto*  
**then have**  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$   
**by** (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(1,3) *mult-zero-right*  
*monoid-mult-class.power2-eq-square* *semiring-class.distrib-left*)  
**hence**  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot h = 0$   
**using** *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)  
**hence** *obs*:  $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$   
**apply**(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(2, 3)  
*Groups.mult-ac*(2, 3) *monoid-mult-class.power2-eq-square* *nat-distrib*(2))  
**thus**  $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$   
**by** (*simp add*: *monoid-mult-class.power2-eq-square*)

**have**  $2 \cdot g \cdot h + (-((g \cdot \tau) + v))^2 = 0$   
**using** *obs by (metis Groups.add-ac(2) power2-minus)*  
**thus**  $2 \cdot g \cdot h + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$   
**by** (*simp add: monoid-mult-class.power2-eq-square*)  
**qed**

**lemma** [*bb-real-arith*]:

**assumes** *invar*:  $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$   
**shows**  $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$   
 $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$  (**is** *?lhs = ?rhs*)

**proof**—

**have** *?lhs* =  $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$   
**by** (*auto simp: algebra-simps semiring-normalization-rules(29)*)  
**also have**  $\dots = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$  (**is**  $\dots = ?middle$ )  
**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**  $\dots = ?middle$   
**by** (*simp add: semiring-normalization-rules(29)*)  
**finally have** *?rhs* = *?middle*.  
**ultimately show** *?thesis* **by** *auto*

**qed**

**lemma** *bouncing-ball-dyn*:  $g < 0 \implies h \geq 0 \implies \text{Hoare}$

$\llbracket \lambda s. s\$1 = h \wedge s\$2 = 0 \rrbracket$   
 (LOOP  
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$   
 $\rrbracket \llbracket \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rrbracket$   
**apply**(*rule H-loopI, rule H-seq[where R= $\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ]*)  
**by** (*auto simp: bb-real-arith*)

— Verified with the flow

**lemma** *local-flow-ball*: *local-flow* (*f g*) UNIV UNIV ( $\varphi \ g$ )

**apply**(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff, clarsimp*)  
**apply**(*rule-tac x=1/2 in exI, clarsimp, rule-tac x=1 in exI*)  
**apply**(*simp add: dist-norm norm-vec-def L2-set-def UNIV-2*)  
**by** (*auto simp: forall-2 intro!: poly-derivatives*)

**lemma** *bouncing-ball-flow*:  $g < 0 \implies h \geq 0 \implies \text{Hoare}$

$\llbracket \lambda s. s\$1 = h \wedge s\$2 = 0 \rrbracket$   
 (LOOP  
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0));$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$   
 $\rrbracket \llbracket \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rrbracket$   
**apply**(*rule H-loopI*)  
**apply**(*rule H-seq[where R= $\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2$ ]*)  
**apply**(*subst local-flow.sH-g-ode-subset[OF local-flow-ball], simp*)  
**apply**(*force simp: bb-real-arith*)  
**by** (*rule H-cond*) (*auto simp: bb-real-arith*)

— Refined with annotated dynamics

**lemma** *R-bb-assign*:  $g < (0::real) \implies 0 \leq h \implies$

$2 ::= (\lambda s. - s\$2) \leq \text{Ref}$

$$\begin{aligned} & [\lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2] \\ & [\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2] \\ & \text{by (rule R-assign-rule, auto)} \end{aligned}$$

**lemma** *R-bouncing-ball-dyn*:

**assumes**  $g < 0$  **and**  $h \geq 0$   
**shows**  $\text{Ref } [\lambda s. s\$1 = h \wedge s\$2 = 0] \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h] \geq$   
 $(\text{LOOP}$   
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$   
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$   
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$   
**apply**(rule order-trans)  
**apply**(rule R-loop-mono) **defer**  
**apply**(rule R-loop)  
**apply**(rule R-seq)  
**using** *assms* **apply**(simp-all, force simp: bb-real-arith)  
**apply**(rule R-seq-mono) **defer**  
**apply**(rule order-trans)  
**apply**(rule R-cond-mono) **defer defer**  
**apply**(rule R-cond) **defer**  
**using** *R-bb-assign* **apply** force  
**apply**(rule R-skip, clarsimp)  
**by** (rule R-g-evol-rule, force simp: bb-real-arith)

**no-notation** *fball* ( $f$ )  
**and** *ball-flow* ( $\varphi$ )

## Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every  $\tau$  minutes, it sets its chronometer to 0, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE  $T' = -a * (T - U)$  where  $U = L \geq 0$  when the heater is on, and  $U = 0$  when it is off. We use  $1$  to denote the room's temperature,  $2$  is time as measured by the thermostat's chronometer, and  $3$  is a variable to save temperature measurements. Finally,  $4$  states whether the heater is on ( $s\$4 = 1$ ) or off ( $s\$4 = 0$ ). We prove that the thermostat keeps the room's temperature between  $Tmin$  and  $Tmax$ .

**abbreviation** *therm-vec-field* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{real}^4$  ( $f$ )  
**where**  $f \ a \ L \ s \equiv (\chi \ i. \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } -a * (s\$1 - L) \text{ else } 0))$

**abbreviation** *therm-guard* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{bool}$  ( $G$ )  
**where**  $G \ Tmin \ Tmax \ a \ L \ s \equiv (s\$2 \leq -(\ln((L - (\text{if } L=0 \text{ then } Tmin \text{ else } Tmax)) / (L - s\$3))) / a)$

**abbreviation** *therm-loop-inv* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{bool}$  ( $I$ )  
**where**  $I \ Tmin \ Tmax \ s \equiv Tmin \leq s\$1 \wedge s\$1 \leq Tmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

**abbreviation** *therm-flow* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{real}^4$  ( $\varphi$ )  
**where**  $\varphi \ a \ L \ \tau \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } -\exp(-a * \tau) * (L - s\$1) + L \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

— Verified with the flow

**lemma** *norm-diff-therm-dyn*:  $0 < a \implies \|f \ a \ L \ s_1 - f \ a \ L \ s_2\| = |a| * |s_1\$1 - s_2\$1|$

**proof**(simp add: norm-vec-def L2-set-def, unfold UNIV-4, simp)

**assume**  $a1: 0 < a$   
**have**  $f2: \bigwedge r \ ra. |(r::\text{real}) + - \ ra| = |ra + - \ r|$   
**by** (metis abs-minus-commute minus-real-def)  
**have**  $\bigwedge r \ ra \ rb. (r::\text{real}) * ra + - \ (r * rb) = r * (ra + - \ rb)$   
**by** (metis minus-real-def right-diff-distrib)

hence  $|a * (s_1\$1 + - L) + - (a * (s_2\$1 + - L))| = a * |s_1\$1 + - s_2\$1|$   
 using *a1* by (*simp add: abs-mult*)  
 thus  $|a * (s_2\$1 - L) - a * (s_1\$1 - L)| = a * |s_1\$1 - s_2\$1|$   
 using *f2 minus-real-def* by *presburger*  
 qed

lemma *local-lipschitz-therm-dyn*:

assumes  $0 < (a::real)$   
 shows *local-lipschitz UNIV UNIV* ( $\lambda t::real. f\ a\ L$ )  
 apply(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)  
 apply(*clarsimp, rule-tac x=1 in exI, clarsimp, rule-tac x=a in exI*)  
 using *assms* apply(*simp-all add: norm-diff-therm-dyn*)  
 apply(*simp add: norm-vec-def L2-set-def, unfold UNIV-4, clarsimp*)  
 unfolding *real-sqrt-abs[symmetric]* by (*rule real-le-lsqr*) *auto*

lemma *local-flow-therm*:  $a > 0 \implies \text{local-flow } (f\ a\ L)\ UNIV\ UNIV\ (\varphi\ a\ L)$   
 by (*unfold-locales, auto intro!: poly-derivatives local-lipschitz-therm-dyn*  
*simp: forall-4 vec-eq-iff*)

lemma *therm-dyn-down-real-arith*:

assumes  $a > 0$  and *Thyps*:  $0 < T_{min}\ T_{min} \leq T\ T \leq T_{max}$   
 and *thyys*:  $0 \leq (\tau::real) \ \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(T_{min} / T) / a)$   
 shows  $T_{min} \leq \exp(-a * \tau) * T$  and  $\exp(-a * \tau) * T \leq T_{max}$

proof—

have  $0 \leq \tau \wedge \tau \leq -(\ln(T_{min} / T) / a)$   
 using *thyys* by *auto*  
 hence  $\ln(T_{min} / T) \leq -a * \tau \wedge -a * \tau \leq 0$   
 using *assms(1) divide-le-cancel* by *fastforce*  
 also have  $T_{min} / T > 0$   
 using *Thyps* by *auto*  
 ultimately have *obs*:  $T_{min} / T \leq \exp(-a * \tau) \exp(-a * \tau) \leq 1$   
 using *exp-ln exp-le-one-iff* by (*metis exp-less-cancel-iff not-less, simp*)  
 thus  $T_{min} \leq \exp(-a * \tau) * T$   
 using *Thyps* by (*simp add: pos-divide-le-eq*)  
 show  $\exp(-a * \tau) * T \leq T_{max}$   
 using *Thyps mult-left-le-one-le[OF - exp-ge-zero obs(2), of T]*  
*less-eq-real-def order-trans-rules(23)* by *blast*

qed

lemma *therm-dyn-up-real-arith*:

assumes  $a > 0$  and *Thyps*:  $T_{min} \leq T\ T \leq T_{max}\ T_{max} < (L::real)$   
 and *thyys*:  $0 \leq \tau \ \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$   
 shows  $L - T_{max} \leq \exp(-(a * \tau)) * (L - T)$   
 and  $L - \exp(-(a * \tau)) * (L - T) \leq T_{max}$   
 and  $T_{min} \leq L - \exp(-(a * \tau)) * (L - T)$

proof—

have  $0 \leq \tau \wedge \tau \leq -(\ln((L - T_{max}) / (L - T)) / a)$   
 using *thyys* by *auto*  
 hence  $\ln((L - T_{max}) / (L - T)) \leq -a * \tau \wedge -a * \tau \leq 0$   
 using *assms(1) divide-le-cancel* by *fastforce*  
 also have  $(L - T_{max}) / (L - T) > 0$   
 using *Thyps* by *auto*  
 ultimately have  $(L - T_{max}) / (L - T) \leq \exp(-a * \tau) \wedge \exp(-a * \tau) \leq 1$   
 using *exp-ln exp-le-one-iff* by (*metis exp-less-cancel-iff not-less*)  
 moreover have  $L - T > 0$   
 using *Thyps* by *auto*  
 ultimately have *obs*:  $(L - T_{max}) \leq \exp(-a * \tau) * (L - T) \wedge \exp(-a * \tau) * (L - T) \leq (L - T)$   
 by (*simp add: pos-divide-le-eq*)  
 thus  $(L - T_{max}) \leq \exp(-(a * \tau)) * (L - T)$   
 by *auto*

```

thus  $L - \exp(-(a * \tau)) * (L - T) \leq Tmax$ 
by auto
show  $Tmin \leq L - \exp(-(a * \tau)) * (L - T)$ 
using Thyps and obs by auto
qed

```

**lemmas** *H-g-ode-therm* = *local-flow.sH-g-ode-ivl*[*OF local-flow-therm* - *UNIV-I*]

**lemma** *thermostat-flow*:

```

assumes  $0 < a$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows Hoare [I Tmin Tmax]
(LOOP (
  — control
  ( $2 ::= (\lambda s. 0)$ );
  ( $3 ::= (\lambda s. s\$1)$ );
  (IF ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq Tmin + 1$ ) THEN
    ( $4 ::= (\lambda s. 1)$ )
    ELSE IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq Tmax - 1$ ) THEN
      ( $4 ::= (\lambda s. 0)$ )
    ELSE skip);
  — dynamics
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = (\lambda t. f\ a\ 0) \ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ )
  ELSE
    ( $x' = (\lambda t. f\ a\ L) \ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ ))
) INV I Tmin Tmax)
[I Tmin Tmax]
apply(rule H-loopI)
apply(rule-tac  $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0$  in H-seq, simp, simp)
apply(rule H-cond, simp-all add: H-g-ode-therm[OF assms(1,2)]) +
using therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin]
and therm-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

```

— Refined with the flow

**lemma** *R-therm-dyn-down*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows Ref [ $\lambda s. s\$4 = 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I Tmin Tmax]  $\geq$ 
  ( $x' = (\lambda t. f\ a\ 0) \ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ )
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-down-real-arith[OF assms(1,3), of - Tmax] by auto

```

**lemma** *R-therm-dyn-up*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows Ref [ $\lambda s. s\$4 \neq 0 \wedge I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I Tmin Tmax]  $\geq$ 
  ( $x' = (\lambda t. f\ a\ L) \ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ )
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-up-real-arith[OF assms(1) - - assms(4), of Tmin] by auto

```

**lemma** *R-therm-dyn*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
shows Ref [ $\lambda s. I\ Tmin\ Tmax\ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I Tmin Tmax]  $\geq$ 
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = (\lambda t. f\ a\ 0) \ \&\ G\ Tmin\ Tmax\ a\ 0\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ )
  ELSE
    ( $x' = (\lambda t. f\ a\ L) \ \&\ G\ Tmin\ Tmax\ a\ L\ on\ (\lambda s. \{0..\tau\})\ UNIV\ @\ 0$ ))
apply(rule order-trans, rule R-cond-mono)
using R-therm-dyn-down[OF assms] R-therm-dyn-up[OF assms] by (auto intro!: R-cond)

```



**lemma** *R-therm-assign1*:  $\text{Ref } [I \text{ Tmin Tmax}] [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0] \geq (2 ::= (\lambda s. 0))$   
**by** (*auto simp*: *R-assign-rule*)

**lemma** *R-therm-assign2*:  
 $\text{Ref } [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0] [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq (3 ::= (\lambda s. s\$1))$   
**by** (*auto simp*: *R-assign-rule*)

**lemma** *R-therm-ctrl*:  
 $\text{Ref } [I \text{ Tmin Tmax}] [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$   
 $(2 ::= (\lambda s. 0));$   
 $(3 ::= (\lambda s. s\$1));$   
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq \text{Tmin} + 1) \text{ THEN}$   
 $(4 ::= (\lambda s. 1))$   
 $\text{ELSE IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{Tmax} - 1) \text{ THEN}$   
 $(4 ::= (\lambda s. 0))$   
 $\text{ELSE skip})$   
**apply**(*rule R-seq-rule*)+  
**apply**(*rule R-therm-assign1*)  
**apply**(*rule R-therm-assign2*)  
**apply**(*rule order-trans*)  
**apply**(*rule R-cond-mono*)  
**apply**(*rule R-assign-rule*) **defer**  
**apply**(*rule R-cond-mono*)  
**apply**(*rule R-assign-rule*) **defer**  
**apply**(*rule R-skip*) **defer**  
**apply**(*rule order-trans*)  
**apply**(*rule R-cond-mono*)  
**apply** *force*  
**by** (*rule R-cond*)+ *auto*

**lemma** *R-therm-loop*:  $\text{Ref } [I \text{ Tmin Tmax}] [I \text{ Tmin Tmax}] \geq$   
 $(\text{LOOP}$   
 $\text{Ref } [I \text{ Tmin Tmax}] [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0 \wedge s\$3 = s\$1];$   
 $\text{Ref } [\lambda s. I \text{ Tmin Tmax } s \wedge s\$2 = 0 \wedge s\$3 = s\$1] [I \text{ Tmin Tmax}]$   
 $\text{INV } I \text{ Tmin Tmax})$   
**by** (*intro R-loop R-seq, simp-all*)

**lemma** *R-thermostat-flow*:  
**assumes**  $a > 0$  **and**  $0 \leq \tau$  **and**  $0 < \text{Tmin}$  **and**  $\text{Tmax} < L$   
**shows**  $\text{Ref } [I \text{ Tmin Tmax}] [I \text{ Tmin Tmax}] \geq$   
 $(\text{LOOP } ($   
 $\text{— control}$   
 $(2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1));$   
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq \text{Tmin} + 1) \text{ THEN}$   
 $(4 ::= (\lambda s. 1))$   
 $\text{ELSE IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{Tmax} - 1) \text{ THEN}$   
 $(4 ::= (\lambda s. 0))$   
 $\text{ELSE skip});$   
 $\text{— dynamics}$   
 $(\text{IF } (\lambda s. s\$4 = 0) \text{ THEN}$   
 $(x' = (\lambda t. f \ a \ 0) \ \& \ G \ \text{Tmin} \ \text{Tmax} \ a \ 0 \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0)$   
 $\text{ELSE}$   
 $(x' = (\lambda t. f \ a \ L) \ \& \ G \ \text{Tmin} \ \text{Tmax} \ a \ L \ \text{on } (\lambda s. \{0..\tau\}) \ \text{UNIV} \ @ \ 0))$   
 $) \ \text{INV } I \text{ Tmin Tmax})$   
**by** (*intro order-trans*[*OF* - *R-therm-loop*] *R-loop-mono*  
*R-seq-mono* *R-therm-ctrl* *R-therm-dyn*[*OF* *assms*])

**no-notation** *therm-vec-field* ( $f$ )  
**and** *therm-flow* ( $\varphi$ )

**and** *therm-guard* ( $G$ )  
**and** *therm-loop-inv* ( $I$ )

## Water tank

— Variation of Hespanha and [?]

**abbreviation** *tank-vec-field* ::  $real \Rightarrow real^4 \Rightarrow real^4$  ( $f$ )  
**where**  $f\ k\ s \equiv (\chi\ i.\ \text{if } i = 2 \text{ then } 1 \text{ else } (\text{if } i = 1 \text{ then } k \text{ else } 0))$

**abbreviation** *tank-flow* ::  $real \Rightarrow real \Rightarrow real^4 \Rightarrow real^4$  ( $\varphi$ )  
**where**  $\varphi\ k\ \tau\ s \equiv (\chi\ i.\ \text{if } i = 1 \text{ then } k * \tau + s\$1 \text{ else } (\text{if } i = 2 \text{ then } \tau + s\$2 \text{ else } s\$i))$

**abbreviation** *tank-guard* ::  $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$  ( $G$ )  
**where**  $G\ Hm\ k\ s \equiv s\$2 \leq (Hm - s\$3)/k$

**abbreviation** *tank-loop-inv* ::  $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$  ( $I$ )  
**where**  $I\ hmin\ hmax\ s \equiv hmin \leq s\$1 \wedge s\$1 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

**abbreviation** *tank-diff-inv* ::  $real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$  ( $dI$ )  
**where**  $dI\ hmin\ hmax\ k\ s \equiv s\$1 = k \cdot s\$2 + s\$3 \wedge 0 \leq s\$2 \wedge hmin \leq s\$3 \wedge s\$3 \leq hmax \wedge (s\$4 = 0 \vee s\$4 = 1)$

— Verified with the flow

**lemma** *local-flow-tank*: *local-flow* ( $f\ k$ ) *UNIV UNIV* ( $\varphi\ k$ )  
**apply** (*unfold-locales*, *unfold local-lipschitz-def lipschitz-on-def*, *simp-all*, *clarsimp*)  
**apply**(*rule-tac*  $x=1/2$  **in** *exI*, *clarsimp*, *rule-tac*  $x=1$  **in** *exI*)  
**apply**(*simp add: dist-norm norm-vec-def L2-set-def*, *unfold UNIV-4*)  
**by** (*auto intro!: poly-derivatives simp: vec-eq-iff*)

**lemma** *tank-arith*:

**assumes**  $0 \leq (\tau::real)$  **and**  $0 < c_o$  **and**  $c_o < c_i$   
**shows**  $\forall \tau \in \{0..\tau\}.\ \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$   
**and**  $\forall \tau \in \{0..\tau\}.\ \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$   
**and**  $hmin \leq y \implies hmin \leq (c_i - c_o) \cdot \tau + y$   
**and**  $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$   
**apply**(*simp-all add: field-simps le-divide-eq assms*)  
**using** *assms* **apply** (*meson add-mono less-eq-real-def mult-left-mono*)  
**using** *assms* **by** (*meson add-increasing2 less-eq-real-def mult-nonneg-nonneg*)

**lemmas** *H-g-ode-tank* = *local-flow.sH-g-ode-ivl*[*OF local-flow-tank - UNIV-I*]

**lemma** *tank-flow*:

**assumes**  $0 \leq \tau$  **and**  $0 < c_o$  **and**  $c_o < c_i$   
**shows** *Hoare* [ $I\ hmin\ hmax$ ]  
(*LOOP*  
— *control*  
 $((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$   
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) \text{ THEN } (4 ::= (\lambda s. 1)) \text{ ELSE}$   
 $(\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) \text{ THEN } (4 ::= (\lambda s. 0)) \text{ ELSE skip}));$   
— *dynamics*  
 $(\text{IF } (\lambda s. s\$4 = 0) \text{ THEN } (x' = (\lambda t. f\ (c_i - c_o)) \ \&\ G\ hmax\ (c_i - c_o) \text{ on } (\lambda s. \{0..\tau\}) \text{ UNIV @ } 0)$   
 $\text{ELSE } (x' = (\lambda t. f\ (-c_o)) \ \&\ G\ hmin\ (-c_o) \text{ on } (\lambda s. \{0..\tau\}) \text{ UNIV @ } 0))$ )  
 $INV\ I\ hmin\ hmax$ ) [ $I\ hmin\ hmax$ ]  
**apply**(*rule H-loopI*)  
**apply**(*rule-tac*  $R=\lambda s. I\ hmin\ hmax\ s \wedge s\$2=0 \wedge s\$3 = s\$1$  **in** *H-seq*)  
**apply**(*rule-tac*  $R=\lambda s. I\ hmin\ hmax\ s \wedge s\$2=0 \wedge s\$3 = s\$1$  **in** *H-seq*)  
**apply**(*rule-tac*  $R=\lambda s. I\ hmin\ hmax\ s \wedge s\$2=0$  **in** *H-seq*, *simp*, *simp*)

```

apply(rule H-cond, simp-all add: H-g-ode-tank[OF assms(1)])
using assms tank-arith[OF - assms(2,3)] by auto

```

— Verified with differential invariants

**lemma** *tank-diff-inv*:

```

0 ≤ τ ⇒ diff-invariant (dI hmin hmax k) (λt. f k) (λs. {0..τ}) UNIV 0 Guard
apply(intro diff-invariant-conj-rule)
  apply(force intro!: poly-derivatives diff-invariant-rules)
  apply(rule-tac ν'=λt. 0 and μ'=λt. 1 in diff-invariant-leq-rule, simp-all, presburger)
  apply(rule-tac ν'=λt. 0 and μ'=λt. 0 in diff-invariant-leq-rule, simp-all)
  apply(force intro!: poly-derivatives)+
by (auto intro!: poly-derivatives diff-invariant-rules)

```

**lemma** *tank-inv-arith1*:

```

assumes 0 ≤ (τ::real) and co < ci and b: hmin ≤ y0 and g: τ ≤ (hmax - y0) / (ci - co)
shows hmin ≤ (ci - co) · τ + y0 and (ci - co) · τ + y0 ≤ hmax

```

**proof**—

```

have (ci - co) · τ ≤ (hmax - y0)
  using g assms(2,3) by (metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq)
thus (ci - co) · τ + y0 ≤ hmax
  by auto
show hmin ≤ (ci - co) · τ + y0
  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 ≤ (τ::real) and 0 < co and b: y0 ≤ hmax and g: τ ≤ - ((hmin - y0) / co)
shows hmin ≤ y0 - co · τ and y0 - co · τ ≤ hmax

```

**proof**—

```

have τ · co ≤ y0 - hmin
  using g ⟨0 < co⟩ pos-le-minus-divide-eq by fastforce
thus hmin ≤ y0 - co · τ
  by (auto simp: mult.commute)
show y0 - co · τ ≤ hmax
  using b assms(1,2) by (smt mult-nonneg-nonneg)

```

**qed**

**lemma** *tank-inv*:

```

assumes 0 ≤ τ and 0 < co and co < ci
shows Hoare [I hmin hmax]

```

(LOOP

— control

```

((2 ::= (λs. 0)); (3 ::= (λs. s$1)));
(IF (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) THEN (4 ::= (λs. 1)) ELSE
  (IF (λs. s$4 = 1 ∧ s$3 ≥ hmax - 1) THEN (4 ::= (λs. 0)) ELSE skip));

```

— dynamics

```

(IF (λs. s$4 = 0) THEN
  (x' = (λt. f (ci - co)) & G hmax (ci - co) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (ci - co)))
  ELSE
  (x' = (λt. f (-co)) & G hmin (-co) on (λs. {0..τ}) UNIV @ 0 DINV (dI hmin hmax (-co))))

```

INV I hmin hmax) [I hmin hmax]

**apply**(rule *H-loopI*)

**apply**(rule-tac R=λs. I hmin hmax s ∧ s\$2=0 ∧ s\$3 = s\$1 **in** *H-seq*)

**apply**(rule-tac R=λs. I hmin hmax s ∧ s\$2=0 ∧ s\$3 = s\$1 **in** *H-seq*)

**apply**(rule-tac R=λs. I hmin hmax s ∧ s\$2=0 **in** *H-seq*, simp, simp)

**apply**(rule *H-cond*, simp, simp)+

**apply**(rule *H-cond*, rule *H-g-ode-inv*)

**using** *assms* *tank-inv-arith1* **apply**(force simp: *tank-diff-inv*, simp, clarsimp)

**apply**(rule *H-g-ode-inv*)  
**using** *assms tank-diff-inv*[*of -c<sub>o</sub> hmin hmax*] *tank-inv-arith2* **by** *auto*

— Refined with differential invariants

**lemma** *R-tank-inv*:

**assumes**  $0 \leq \tau$  **and**  $0 < c_o$  **and**  $c_o < c_i$   
**shows** *Ref* [*I hmin hmax*] [*I hmin hmax*]  $\geq$   
(*LOOP*

— control

(( $2 ::= (\lambda s. 0)$ );( $3 ::= (\lambda s. s\$1)$ );  
(*IF* ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ ) *THEN* ( $4 ::= (\lambda s. 1)$ ) *ELSE*  
(*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) *THEN* ( $4 ::= (\lambda s. 0)$ ) *ELSE skip*));

— dynamics

(*IF* ( $\lambda s. s\$4 = 0$ ) *THEN*

( $x' = (\lambda t. f(c_i - c_o)) \ \& \ G \ hmax(c_i - c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax \ (c_i - c_o))$ )  
*ELSE*

( $x' = (\lambda t. f(-c_o)) \ \& \ G \ hmin(-c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax \ (-c_o))$ )) )

*INV I hmin hmax*) (**is** *LOOP* (*?ctrl*; *?dyn*) *INV -*  $\leq$  *?ref*)

**proof**—

— First we refine the control.

**let** *?Icntrl* =  $\lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$

**and** *?cond* =  $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$

**have** *ifbranch1*:  $4 ::= (\lambda s. 1) \leq Ref \ [\lambda s. ?cond \ s \wedge ?Icntrl \ s] \ [?Icntrl] \ (\mathbf{is} \ - \leq ?branch1)$

**by** (rule *R-assign-rule*, *simp*)

**have** *ifbranch2*: (*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) *THEN* ( $4 ::= (\lambda s. 0)$ ) *ELSE skip*)  $\leq$

*Ref* [ $\lambda s. \neg ?cond \ s \wedge ?Icntrl \ s$ ] [*?Icntrl*] (**is**  $- \leq ?branch2$ )

**apply**(rule *order-trans*, rule *R-cond-mono*) **defer defer**

**by** (rule *R-cond*) (*auto intro!*: *R-assign-rule R-skip*)

**have** *ifthenelse*: (*IF* *?cond* *THEN* *?branch1* *ELSE* *?branch2*)  $\leq Ref \ [?Icntrl] \ [?Icntrl] \ (\mathbf{is} \ ?ifthenelse \leq$

-)

**by** (rule *R-cond*)

**have** (*IF* *?cond* *THEN* ( $4 ::= (\lambda s. 1)$ ) *ELSE* (*IF* ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) *THEN* ( $4 ::= (\lambda s. 0)$ ) *ELSE skip*))  $\leq$

*Ref* [*?Icntrl*] [*?Icntrl*]

**apply**(rule-tac *y=?ifthenelse* **in** *order-trans*, rule *R-cond-mono*)

**using** *ifbranch1 ifbranch2 ifthenelse* **by** *auto*

**hence** *ctrl*: *?ctrl*  $\leq Ref \ [I \ hmin \ hmax] \ [?Icntrl]$

**apply**(rule-tac *R=?Icntrl* **in** *R-seq-rule*)

**apply**(rule-tac *R=?Icntrl* **in** *R-seq-rule*)

**by** (*auto intro!*: *R-assign-rule*)

— Then we refine the dynamics.

**have** *dynup*: ( $x' = (\lambda t. f(c_i - c_o)) \ \& \ G \ hmax(c_i - c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax \ (c_i - c_o))$ )  $\leq$

*Ref* [ $\lambda s. s\$4 = 0 \wedge ?Icntrl \ s$ ] [*I hmin hmax*]

**apply**(rule *R-g-ode-inv*[*OF tank-diff-inv*[*OF assms*(1)])])

**using** *assms* **by** (*auto simp*: *tank-inv-arith1*)

**have** *dyndown*: ( $x' = (\lambda t. f(-c_o)) \ \& \ G \ hmin(-c_o) \ on \ (\lambda s. \{0.. \tau\}) \ UNIV \ @ \ 0 \ DINV \ (dI \ hmin \ hmax \ (-c_o))$ )  $\leq$

*Ref* [ $\lambda s. s\$4 \neq 0 \wedge ?Icntrl \ s$ ] [*I hmin hmax*]

**apply**(rule *R-g-ode-inv*)

**using** *tank-diff-inv*[*OF assms*(1), *of -c<sub>o</sub>*] *assms*

**by** (*auto simp*: *tank-inv-arith2*)

**have** *dyn*: *?dyn*  $\leq Ref \ [?Icntrl] \ [I \ hmin \ hmax]$

**apply**(rule *order-trans*, rule *R-cond-mono*)

**using** *dynup dyndown* **by** (*auto intro!*: *R-cond*)

— Finally we put everything together.

**have** *pre-pos*: [*I hmin hmax*]  $\leq [I \ hmin \ hmax]$

**by** *simp*

**have** *inv-inv*: *Ref* [*I hmin hmax*] [*?Icntrl*]; (*Ref* [*?Icntrl*] [*I hmin hmax*])  $\leq Ref \ [I \ hmin \ hmax] \ [I \ hmin$

```

hmax]
  by (rule R-seq)
  have loopref: LOOP Ref [I hmin hmax] [?Ictrl]; (Ref [?Ictrl] [I hmin hmax]) INV I hmin hmax ≤
  ?ref
  apply(rule R-loop)
  using pre-pos inv-inv by auto
  have obs: ?ctrl;?dyn ≤ Ref [I hmin hmax] [?Ictrl]; (Ref [?Ictrl] [I hmin hmax])
  apply(rule R-seq-mono)
  using ctrl dyn by auto
  show LOOP (?ctrl;?dyn) INV I hmin hmax ≤ ?ref
  by (rule order-trans[OF - loopref], rule R-loop-mono[OF obs])
qed

no-notation tank-vec-field (f)
  and tank-flow (φ)
  and tank-guard (G)
  and tank-loop-inv (I)
  and tank-diff-inv (dI)

end

```

## 0.11 Mathematical Preliminaries

This section adds useful syntax, abbreviations and theorems to the Isabelle distribution.

```

theory MTX-Preliminaries
  imports ../HS-Preliminaries

```

```
begin
```

### 0.11.1 Syntax

```
abbreviation e k ≡ axis k 1
```

```
syntax
```

```
-ivl-integral :: real ⇒ real ⇒ 'a ⇒ pttrn ⇒ bool ((∫ - (-) ∂ / -) [0, 0, 10] 10)
```

```
translations
```

```
∫ab f ∂x ⇒ CONST ivl-integral a b (λx. f)
```

```
notation matrix-inv (--1 [90])
```

```
abbreviation entries (A::'an × 'm) ≡ {A $ i $ j | i j. i ∈ UNIV ∧ j ∈ UNIV}
```

### 0.11.2 Topology and sets

```
lemmas compact-imp-bdd-above = compact-imp-bounded[THEN bounded-imp-bdd-above]
```

```
lemma comp-cont-image-spec: continuous-on T f ⇒ compact T ⇒ compact {f t | t. t ∈ T}
  using compact-continuous-image by (simp add: Setcompr-eq-image)
```

```
lemmas bdd-above-cont-comp-spec = compact-imp-bdd-above[OF comp-cont-image-spec]
```

```
lemmas bdd-above-norm-cont-comp = continuous-on-norm[THEN bdd-above-cont-comp-spec]
```

```
lemma open-cballE: t0 ∈ T ⇒ open T ⇒ ∃ e>0. cball t0 e ⊆ T
  using open-contains-cball by blast
```

```
lemma open-ballE: t0 ∈ T ⇒ open T ⇒ ∃ e>0. ball t0 e ⊆ T
  using open-contains-ball by blast
```

**lemma** *funcset-UNIV*:  $f \in A \rightarrow UNIV$   
**by** *auto*

**lemma** *finite-image-of-finite[simp]*:  
**fixes**  $f :: 'a :: finite \Rightarrow 'b$   
**shows** *finite*  $\{x. \exists i. x = f\ i\}$   
**using** *finite-Atleast-Atmost-nat* **by** *force*

**lemma** *finite-image-of-finite2*:  
**fixes**  $f :: 'a :: finite \Rightarrow 'b :: finite \Rightarrow 'c$   
**shows** *finite*  $\{f\ x\ y \mid x\ y. P\ x\ y\}$

**proof**—  
**have** *finite*  $(\bigcup x. \{f\ x\ y \mid y. P\ x\ y\})$   
**by** *simp*  
**moreover have**  $\{f\ x\ y \mid x\ y. P\ x\ y\} = (\bigcup x. \{f\ x\ y \mid y. P\ x\ y\})$   
**by** *auto*  
**ultimately show** *?thesis*  
**by** *simp*  
**qed**

### 0.11.3 Functions

**lemma** *finite-sum-univ-singleton*:  $(\text{sum } g\ UNIV) = \text{sum } g\ \{i :: 'a :: finite\} + \text{sum } g\ (UNIV - \{i\})$   
**by** (*metis add.commute finite-class.finite-UNIV sum.subset-diff top-greatest*)

**lemma** *suminfI*:  
**fixes**  $f :: nat \Rightarrow 'a :: \{t2\text{-space}, \text{comm-monoid-add}\}$   
**shows**  $f\ \text{sums } k \implies \text{suminf } f = k$   
**unfolding** *sums-iff* **by** *simp*

**lemma** *suminf-eq-sum*:  
**fixes**  $f :: nat \Rightarrow ('a :: \text{real-normed-vector})$   
**assumes**  $\bigwedge n. n > m \implies f\ n = 0$   
**shows**  $(\sum n. f\ n) = (\sum n \leq m. f\ n)$   
**using** *assms* **by** (*meson atMost-iff finite-atMost not-le suminf-finite*)

**lemma** *suminf-mult*:  $\text{summable } f \implies (\sum n. f\ n * c) = (\sum n. f\ n) * c$  **for**  $c :: 'a :: \text{real-normed-algebra}$   
**by** (*rule bounded-linear.suminf [OF bounded-linear-mult-left, symmetric]*)

**lemma** *sum-if-then-else-simps[simp]*:  
**fixes**  $q :: ('a :: \text{semiring-0})$  **and**  $i :: 'n :: finite$   
**shows**  $(\sum j \in UNIV. f\ j * (\text{if } j = i \text{ then } q \text{ else } 0)) = f\ i * q$   
**and**  $(\sum j \in UNIV. f\ j * (\text{if } i = j \text{ then } q \text{ else } 0)) = f\ i * q$   
**and**  $(\sum j \in UNIV. (\text{if } i = j \text{ then } q \text{ else } 0) * f\ j) = q * f\ i$   
**and**  $(\sum j \in UNIV. (\text{if } j = i \text{ then } q \text{ else } 0) * f\ j) = q * f\ i$   
**by** (*auto simp: finite-sum-univ-singleton[of - i]*)

### 0.11.4 Suprema

**lemma** *le-max-image-of-finite[simp]*:  
**fixes**  $f :: 'a :: finite \Rightarrow 'b :: \text{linorder}$   
**shows**  $(f\ i) \leq \text{Max } \{x. \exists i. x = f\ i\}$   
**by** (*rule Max.coboundedI, simp-all*) (*rule-tac x=i in exI, simp*)

**lemma** *cSup-eq*:  
**fixes**  $c :: 'a :: \text{conditionally-complete-lattice}$   
**assumes**  $\forall x \in X. x \leq c$  **and**  $\exists x \in X. c \leq x$   
**shows**  $\text{Sup } X = c$   
**by** (*metis assms cSup-eq-maximum order-class.order.antisym*)

**lemma** *cSup-mem-eq*:

$c \in X \implies \forall x \in X. x \leq c \implies \text{Sup } X = c$  **for**  $c::'a::\text{conditionally-complete-lattice}$   
**by** (*rule cSup-eq, auto*)

**lemma** *cSup-finite-ex*:

$\text{finite } X \implies X \neq \{\} \implies \exists x \in X. \text{Sup } X = x$  **for**  $X::'a::\text{conditionally-complete-linorder set}$   
**by** (*metis (full-types) bdd-finite(1) cSup-upper finite-Sup-less-iff order-less-le*)

**lemma** *cMax-finite-ex*:

$\text{finite } X \implies X \neq \{\} \implies \exists x \in X. \text{Max } X = x$  **for**  $X::'a::\text{conditionally-complete-linorder set}$   
**apply** (*subst cSup-eq-Max[symmetric]*)  
**using** *cSup-finite-ex* **by** *auto*

**lemma** *finite-nat-minimal-witness*:

**fixes**  $P :: ('a::\text{finite}) \Rightarrow \text{nat} \Rightarrow \text{bool}$   
**assumes**  $\forall i. \exists N::\text{nat}. \forall n \geq N. P \ i \ n$   
**shows**  $\exists N. \forall i. \forall n \geq N. P \ i \ n$

**proof**—

**let**  $?bound \ i = (\text{LEAST } N. \forall n \geq N. P \ i \ n)$   
**let**  $?N = \text{Max } \{?bound \ i \mid i. i \in \text{UNIV}\}$   
**{fix**  $n::\text{nat}$  **and**  $i::'a$   
**assume**  $n \geq ?N$   
**obtain**  $M$  **where**  $\forall n \geq M. P \ i \ n$   
**using** *assms* **by** *blast*  
**hence** *obs*:  $\forall m \geq ?bound \ i. P \ i \ m$   
**using** *LeastI*[*of*  $\lambda N. \forall n \geq N. P \ i \ n$ ] **by** *blast*  
**have** *finite*  $\{?bound \ i \mid i. i \in \text{UNIV}\}$   
**by** *simp*  
**hence**  $?N \geq ?bound \ i$   
**using** *Max-ge* **by** *blast*  
**hence**  $n \geq ?bound \ i$   
**using**  $\langle n \geq ?N \rangle$  **by** *linarith*  
**hence**  $P \ i \ n$   
**using** *obs* **by** *blast*  
**thus**  $\exists N. \forall i \ n. N \leq n \longrightarrow P \ i \ n$   
**by** *blast*

**qed**

### 0.11.5 Real numbers

**named-theorems** *field-power-simps* *simplification rules for powers to the nth*

**declare** *semiring-normalization-rules*(18) [*field-power-simps*]  
**and** *semiring-normalization-rules*(26) [*field-power-simps*]  
**and** *semiring-normalization-rules*(27) [*field-power-simps*]  
**and** *semiring-normalization-rules*(28) [*field-power-simps*]  
**and** *semiring-normalization-rules*(29) [*field-power-simps*]

WARNING: Adding  $?x * ?x^{?q} = ?x^{\text{Suc } ?q}$  to our tactic makes its combination with *simp* to loop infinitely in some proofs.

**lemma** *sq-le-cancel*:

**shows**  $(a::\text{real}) \geq 0 \implies b \geq 0 \implies a^2 \leq b * a \implies a \leq b$   
**and**  $(a::\text{real}) \geq 0 \implies b \geq 0 \implies a^2 \leq a * b \implies a \leq b$   
**apply** (*metis less-eq-real-def mult.commute mult-le-cancel-left semiring-normalization-rules(29)*)  
**by** (*metis less-eq-real-def mult-le-cancel-left semiring-normalization-rules(29)*)

**lemma** *frac-diff-eq1*:  $a \neq b \implies a / (a - b) - b / (a - b) = 1$  **for**  $a::\text{real}$

**by** (*metis (no-types, hide-lams) ab-left-minus add.commute add-left-cancel diff-divide-distrib diff-minus-eq-add div-self*)

**lemma** *exp-add*:  $x * y - y * x = 0 \implies \exp (x + y) = \exp x * \exp y$   
**by** (*rule exp-add-commuting*) (*simp add: ac-simps*)

**lemmas** *mult-exp-exp* = *exp-add*[*symmetric*]

### 0.11.6 Vectors and matrices

**lemma** *sum-axis*[*simp*]:  
**fixes**  $q :: ('a::\text{semiring-0})$   
**shows**  $(\sum_{j \in \text{UNIV}} f j * \text{axis } i \ \$ \ j) = f i * q$   
**and**  $(\sum_{j \in \text{UNIV}} \text{axis } i \ \$ \ j * f j) = q * f i$   
**unfolding** *axis-def* **by**(*auto simp: vec-eq-iff*)

**lemma** *sum-scalar-nth-axis*:  $\text{sum } (\lambda i. (x \ \$ \ i) * s \ e \ i) \ \text{UNIV} = x$  **for**  $x :: ('a::\text{semiring-1})^{n^{'}}$   
**unfolding** *vec-eq-iff axis-def* **by** *simp*

**lemma** *scalar-eq-scaleR*[*simp*]:  $c * s \ x = c *_{\text{R}} x$   
**unfolding** *vec-eq-iff* **by** *simp*

**lemma** *matrix-add-rdistrib*:  $((B + C) ** A) = (B ** A) + (C ** A)$   
**by** (*vector matrix-matrix-mult-def sum.distrib*[*symmetric*] *field-simps*)

**lemma** *vec-mult-inner*:  $(A * v) \cdot w = v \cdot (\text{transpose } A * v \ w)$  **for**  $A :: \text{real}^{n^{'}} n^{'}$   
**unfolding** *matrix-vector-mult-def transpose-def inner-vec-def*  
**apply**(*simp add: sum-distrib-right sum-distrib-left*)  
**apply**(*subst sum.swap*)  
**apply**(*subgoal-tac*  $\forall i \ j. A \ \$ \ i \ \$ \ j * v \ \$ \ j * w \ \$ \ i = v \ \$ \ j * (A \ \$ \ i \ \$ \ j * w \ \$ \ i)$ )  
**by** *presburger simp*

**lemma** *uminus-axis-eq*[*simp*]:  $-\text{axis } i \ k = \text{axis } i \ (-k)$  **for**  $k :: 'a::\text{ring}$   
**unfolding** *axis-def* **by**(*simp add: vec-eq-iff*)

**lemma** *norm-axis-eq*[*simp*]:  $\|\text{axis } i \ k\| = \|k\|$

**proof**(*simp add: axis-def norm-vec-def L2-set-def*)

**let**  $? \delta_K = \lambda i \ j \ k. \text{if } i = j \text{ then } k \text{ else } 0$   
**have**  $(\sum_{j \in \text{UNIV}} (\|(? \delta_K \ j \ i \ k)\|)^2) = (\sum_{j \in \{i\}} (\|(? \delta_K \ j \ i \ k)\|)^2) + (\sum_{j \in (\text{UNIV} - \{i\})} (\|(? \delta_K \ j \ i \ k)\|)^2)$   
**using** *finite-sum-univ-singleton* **by** *blast*  
**also have**  $\dots = (\|k\|)^2$   
**by** *simp*  
**finally show**  $\text{sqrt } (\sum_{j \in \text{UNIV}} (\text{norm } (\text{if } j = i \text{ then } k \text{ else } 0))^2) = \text{norm } k$   
**by** *simp*

**qed**

**lemma** *matrix-axis-0*:

**fixes**  $A :: ('a::\text{idom})^{n^{'}} n^{'}$   
**assumes**  $k \neq 0$  **and**  $h: \forall i. (A * v \ (\text{axis } i \ k)) = 0$   
**shows**  $A = 0$

**proof**—

**{fix**  $i :: 'n$   
**have**  $0 = (\sum_{j \in \text{UNIV}} (\text{axis } i \ k) \ \$ \ j * s \ \text{column } j \ A)$   
**using** *h matrix-mult-sum*[*of A axis i k*] **by** *simp*  
**also have**  $\dots = k * s \ \text{column } i \ A$   
**by** (*simp add: axis-def vector-scalar-mult-def column-def vec-eq-iff mult.commute*)  
**finally have**  $k * s \ \text{column } i \ A = 0$   
**unfolding** *axis-def* **by** *simp*  
**hence**  $\text{column } i \ A = 0$   
**using** *vector-mul-eq-0*  $\langle k \neq 0 \rangle$  **by** *blast*}

**thus**  $A = 0$



**unfolding** *column-def vec-eq-iff* **by** *simp*  
**qed**

**lemma** *scaleR-norm-sgn-eq*:  $(\|x\|) *_R \text{sgn } x = x$   
**by** (*metis divideR-right norm-eq-zero scale-eq-0-iff sgn-div-norm*)

**lemma** *vector-scaleR-commute*:  $A *_v c *_R x = c *_R (A *_v x)$  **for**  $x :: ('a::\text{real-normed-algebra-1})^n$   
**unfolding** *scaleR-vec-def matrix-vector-mult-def* **by**(*auto simp: vec-eq-iff scaleR-right.sum*)

**lemma** *scaleR-vector-assoc*:  $c *_R (A *_v x) = (c *_R A) *_v x$  **for**  $x :: ('a::\text{real-normed-algebra-1})^n$   
**unfolding** *matrix-vector-mult-def* **by**(*auto simp: vec-eq-iff scaleR-right.sum*)

**lemma** *mult-norm-matrix-sgn-eq*:  
**fixes**  $x :: ('a::\text{real-normed-algebra-1})^n$   
**shows**  $(\|A *_v \text{sgn } x\|) * (\|x\|) = \|A *_v x\|$

**proof**–

**have**  $\|A *_v x\| = \|A *_v ((\|x\|) *_R \text{sgn } x)\|$   
**by**(*simp add: scaleR-norm-sgn-eq*)  
**also have**  $\dots = (\|A *_v \text{sgn } x\|) * (\|x\|)$   
**by**(*simp add: vector-scaleR-commute*)  
**finally show** *?thesis* ..

**qed**

### 0.11.7 Diagonalization

**lemma** *invertibleI*:  $A ** B = \text{mat } 1 \implies B ** A = \text{mat } 1 \implies \text{invertible } A$   
**unfolding** *invertible-def* **by** *auto*

**lemma** *invertibleD[simp]*:  
**assumes** *invertible A*  
**shows**  $A^{-1} ** A = \text{mat } 1$  **and**  $A ** A^{-1} = \text{mat } 1$   
**using** *assms unfolding matrix-inv-def invertible-def*  
**by** (*simp-all add: verit-sko-ex'*)

**lemma** *matrix-inv-unique*:  
**assumes**  $A ** B = \text{mat } 1$  **and**  $B ** A = \text{mat } 1$   
**shows**  $A^{-1} = B$   
**by** (*metis assms invertibleD(2) invertibleI matrix-mul-assoc matrix-mul-lid*)

**lemma** *invertible-matrix-inv*:  $\text{invertible } A \implies \text{invertible } (A^{-1})$   
**using** *invertibleD invertibleI* **by** *blast*

**lemma** *matrix-inv-idempotent[simp]*:  $\text{invertible } A \implies A^{-1-1} = A$   
**using** *invertibleD matrix-inv-unique* **by** *blast*

**lemma** *matrix-inv-matrix-mul*:  
**assumes** *invertible A* **and** *invertible B*  
**shows**  $(A ** B)^{-1} = B^{-1} ** A^{-1}$

**proof**(*rule matrix-inv-unique*)

**have**  $A ** B ** (B^{-1} ** A^{-1}) = A ** (B ** B^{-1}) ** A^{-1}$   
**by** (*simp add: matrix-mul-assoc*)  
**also have**  $\dots = \text{mat } 1$   
**using** *assms* **by** *simp*  
**finally show**  $A ** B ** (B^{-1} ** A^{-1}) = \text{mat } 1$  .

**next**

**have**  $B^{-1} ** A^{-1} ** (A ** B) = B^{-1} ** (A^{-1} ** A) ** B$   
**by** (*simp add: matrix-mul-assoc*)  
**also have**  $\dots = \text{mat } 1$   
**using** *assms* **by** *simp*  
**finally show**  $B^{-1} ** A^{-1} ** (A ** B) = \text{mat } 1$  .

qed

**lemma** *mat-inverse-simps*[simp]:  
**fixes**  $c :: 'a::\text{division-ring}$   
**assumes**  $c \neq 0$   
**shows**  $\text{mat } (\text{inverse } c) ** \text{mat } c = \text{mat } 1$   
**and**  $\text{mat } c ** \text{mat } (\text{inverse } c) = \text{mat } 1$   
**unfolding** *matrix-matrix-mult-def mat-def* **by** (*auto simp: vec-eq-iff assms*)

**lemma** *matrix-inv-mat*[simp]:  $c \neq 0 \implies (\text{mat } c)^{-1} = \text{mat } (\text{inverse } c)$  **for**  $c :: 'a::\text{division-ring}$   
**by** (*simp add: matrix-inv-unique*)

**lemma** *invertible-mat*[simp]:  $c \neq 0 \implies \text{invertible } (\text{mat } c)$  **for**  $c :: 'a::\text{division-ring}$   
**using** *invertibleI mat-inverse-simps(1) mat-inverse-simps(2)* **by** *blast*

**lemma** *matrix-inv-mat-1*:  $(\text{mat } (1 :: 'a::\text{division-ring}))^{-1} = \text{mat } 1$   
**by** *simp*

**lemma** *invertible-mat-1*:  $\text{invertible } (\text{mat } (1 :: 'a::\text{division-ring}))$   
**by** *simp*

**definition** *similar-matrix* ::  $('a::\text{semiring-1})^m \Rightarrow ('a::\text{semiring-1})^n \Rightarrow \text{bool}$  (**infixr**  $\sim$  25)  
**where** *similar-matrix*  $A B \longleftrightarrow (\exists P. \text{invertible } P \wedge A = P^{-1} ** B ** P)$

**lemma** *similar-matrix-refl*[simp]:  $A \sim A$  **for**  $A :: 'a::\text{division-ring}^n$   
**by** (*unfold similar-matrix-def, rule-tac x=mat 1 in exI, simp*)

**lemma** *similar-matrix-simm*:  $A \sim B \implies B \sim A$  **for**  $A B :: ('a::\text{semiring-1})^n$   
**apply**(*unfold similar-matrix-def, clarsimp*)  
**apply**(*rule-tac x=P^{-1} in exI, simp add: invertible-matrix-inv*)  
**by** (*metis invertible-def matrix-inv-unique matrix-mul-assoc matrix-mul-lid matrix-mul-rid*)

**lemma** *similar-matrix-trans*:  $A \sim B \implies B \sim C \implies A \sim C$  **for**  $A B C :: ('a::\text{semiring-1})^n$

**proof**(*unfold similar-matrix-def, clarsimp*)  
**fix**  $P Q$   
**assume**  $A = P^{-1} ** (Q^{-1} ** C ** Q) ** P$  **and**  $B = Q^{-1} ** C ** Q$   
**let**  $?R = Q ** P$   
**assume** *inverts: invertible Q invertible P*  
**hence**  $?R^{-1} = P^{-1} ** Q^{-1}$   
**by** (*rule matrix-inv-matrix-mul*)  
**also have** *invertible ?R*  
**using** *inverts invertible-mult* **by** *blast*  
**ultimately show**  $\exists R. \text{invertible } R \wedge P^{-1} ** (Q^{-1} ** C ** Q) ** P = R^{-1} ** C ** R$   
**by** (*metis matrix-mul-assoc*)

qed

**lemma** *mat-vec-nth-simps*[simp]:  
 $i = j \implies \text{mat } c \$ i \$ j = c$   
 $i \neq j \implies \text{mat } c \$ i \$ j = 0$   
**by** (*simp-all add: mat-def*)

**definition** *diag-mat*  $f = (\chi \ i \ j. \text{if } i = j \text{ then } f \ i \text{ else } 0)$

**lemma** *diag-mat-vec-nth-simps*[simp]:  
 $i = j \implies \text{diag-mat } f \$ i \$ j = f \ i$   
 $i \neq j \implies \text{diag-mat } f \$ i \$ j = 0$   
**unfolding** *diag-mat-def* **by** *simp-all*

**lemma** *diag-mat-const-eq*[simp]:  $\text{diag-mat } (\lambda i. c) = \text{mat } c$   
**unfolding** *mat-def diag-mat-def* **by** *simp*

**lemma** *matrix-vector-mul-diag-mat*:  $\text{diag-mat } f * v \text{ } s = (\chi \text{ } i. f \text{ } i * s\$i)$   
**unfolding** *diag-mat-def matrix-vector-mult-def* **by** *simp*

**lemma** *matrix-vector-mul-diag-axis*[*simp*]:  $\text{diag-mat } f * v \text{ } (\text{axis } i \text{ } k) = \text{axis } i \text{ } (f \text{ } i * k)$   
**by** (*simp add: matrix-vector-mul-diag-mat axis-def fun-eq-iff*)

**lemma** *matrix-mul-diag-matl*:  $\text{diag-mat } f ** A = (\chi \text{ } i \text{ } j. f \text{ } i * A\$i\$j)$   
**unfolding** *diag-mat-def matrix-matrix-mult-def* **by** *simp*

**lemma** *matrix-matrix-mul-diag-matr*:  $A ** \text{diag-mat } f = (\chi \text{ } i \text{ } j. A\$i\$j * f \text{ } j)$   
**unfolding** *diag-mat-def matrix-matrix-mult-def* **apply**(*clarsimp simp: fun-eq-iff*)  
**subgoal for**  $i \text{ } j$   
**by** (*auto simp: finite-sum-univ-singleton[of - j]*)  
**done**

**lemma** *matrix-mul-diag-diag*:  $\text{diag-mat } f ** \text{diag-mat } g = \text{diag-mat } (\lambda i. f \text{ } i * g \text{ } i)$   
**unfolding** *diag-mat-def matrix-matrix-mult-def vec-eq-iff* **by** *simp*

**lemma** *compow-matrix-mul-diag-mat-eq*:  $((**) (\text{diag-mat } f) ^\wedge n) (\text{mat } 1) = \text{diag-mat } (\lambda i. f \text{ } i ^\wedge n)$   
**apply**(*induct n, simp-all add: matrix-mul-diag-matl*)  
**by** (*auto simp: vec-eq-iff diag-mat-def*)

**lemma** *compow-similar-diag-mat-eq*:  
**assumes** *invertible P*  
**and**  $A = P^{-1} ** (\text{diag-mat } f) ** P$   
**shows**  $((**) A ^\wedge n) (\text{mat } 1) = P^{-1} ** (\text{diag-mat } (\lambda i. f \text{ } i ^\wedge n)) ** P$   
**proof**(*induct n, simp-all add: assms*)  
**fix**  $n::\text{nat}$   
**have**  $P^{-1} ** \text{diag-mat } f ** P ** (P^{-1} ** \text{diag-mat } (\lambda i. f \text{ } i ^\wedge n) ** P) =$   
 $P^{-1} ** \text{diag-mat } f ** \text{diag-mat } (\lambda i. f \text{ } i ^\wedge n) ** P$  **(is ?lhs = -)**  
**by** (*metis (no-types, lifting) assms(1) invertibleD(2) matrix-mul-rid matrix-mul-assoc*)  
**also have**  $\dots = P^{-1} ** \text{diag-mat } (\lambda i. f \text{ } i * f \text{ } i ^\wedge n) ** P$  **(is - = ?rhs)**  
**by** (*metis (full-types) matrix-mul-assoc matrix-mul-diag-diag*)  
**finally show**  $?lhs = ?rhs$  .

**qed**

**lemma** *compow-similar-diag-mat*:  
**assumes**  $A \sim (\text{diag-mat } f)$   
**shows**  $((**) A ^\wedge n) (\text{mat } 1) \sim \text{diag-mat } (\lambda i. f \text{ } i ^\wedge n)$   
**proof**(*unfold similar-matrix-def*)  
**obtain**  $P$  **where** *invertible P* **and**  $A = P^{-1} ** (\text{diag-mat } f) ** P$   
**using** *assms* **unfolding** *similar-matrix-def* **by** *blast*  
**thus**  $\exists P. \text{invertible } P \wedge ((**) A ^\wedge n) (\text{mat } 1) = P^{-1} ** \text{diag-mat } (\lambda i. f \text{ } i ^\wedge n) ** P$   
**using** *compow-similar-diag-mat-eq* **by** *blast*

**qed**

**no-notation** *matrix-inv*  $(^{-1} [90])$   
**and** *similar-matrix* (**infixr**  $\sim$  25)

**end**

## 0.12 Matrix norms

Here, we explore some properties about the operator and the maximum norms for matrices.

**theory** *MTX-Norms*  
**imports** *MTX-Preliminaries*

begin

### 0.12.1 Matrix operator norm

**abbreviation**  $op\text{-}norm :: ('a::real\text{-}normed\text{-}algebra\text{-}1) \wedge 'n \wedge 'm \Rightarrow real ((1||-\|_{op}) [65] 61)$   
**where**  $\|A\|_{op} \equiv onorm (\lambda x. A * v x)$

**lemma** *norm-matrix-bound*:

**fixes**  $A :: ('a::real\text{-}normed\text{-}algebra\text{-}1) \wedge 'n \wedge 'm$   
**shows**  $\|x\| = 1 \implies \|A * v x\| \leq \|(\chi \ i \ j. \|A \$ i \$ j\|) * v \ 1\|$

**proof**—

**fix**  $x :: ('a, 'n) \text{vec}$  **assume**  $\|x\| = 1$   
**hence**  $xi\text{-}le1: \bigwedge i. \|x \$ i\| \leq 1$   
**by** (*metis Finite-Cartesian-Product.norm-nth-le*)  
**{fix**  $j::'m$   
**have**  $\|(\sum i \in UNIV. A \$ j \$ i * x \$ i)\| \leq (\sum i \in UNIV. \|A \$ j \$ i * x \$ i\|)$   
**using** *norm-sum* **by** *blast*  
**also have**  $\dots \leq (\sum i \in UNIV. (\|A \$ j \$ i\|) * (\|x \$ i\|))$   
**by** (*simp add: norm-mult-ineq sum-mono*)  
**also have**  $\dots \leq (\sum i \in UNIV. (\|A \$ j \$ i\|) * 1)$   
**using**  $xi\text{-}le1$  **by** (*simp add: sum-mono mult-left-le*)  
**finally have**  $\|(\sum i \in UNIV. A \$ j \$ i * x \$ i)\| \leq (\sum i \in UNIV. (\|A \$ j \$ i\|) * 1)$  **by** *simp*  
**hence**  $\bigwedge j. \|(A * v x) \$ j\| \leq ((\chi \ i1 \ i2. \|A \$ i1 \$ i2\|) * v \ 1) \$ j$   
**unfolding** *matrix-vector-mult-def* **by** *simp*  
**hence**  $(\sum j \in UNIV. (\|(A * v x) \$ j\|)^2) \leq (\sum j \in UNIV. (((\chi \ i1 \ i2. \|A \$ i1 \$ i2\|) * v \ 1) \$ j)^2)$   
**by** (*metis (mono-tags, lifting) norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)  
**thus**  $\|A * v x\| \leq \|(\chi \ i \ j. \|A \$ i \$ j\|) * v \ 1\|$   
**unfolding** *norm-vec-def L2-set-def* **by** *simp*

qed

**lemma** *onorm-set-proptys*:

**fixes**  $A :: ('a::real\text{-}normed\text{-}algebra\text{-}1) \wedge 'n \wedge 'm$   
**shows** *bounded* (*range* ( $\lambda x. (\|A * v x\|) / (\|x\|)$ ))  
**and** *bdd-above* (*range* ( $\lambda x. (\|A * v x\|) / (\|x\|)$ ))  
**and** (*range* ( $\lambda x. (\|A * v x\|) / (\|x\|)$ ))  $\neq \{\}$   
**unfolding** *bounded-def bdd-above-def image-def dist-real-def*  
**apply**(*rule-tac*  $x=0$  **in**  $exI$ )  
**by** (*rule-tac*  $x=\|(\chi \ i \ j. \|A \$ i \$ j\|) * v \ 1\|$  **in**  $exI$ , *clarsimp*,  
*subst mult-norm-matrix-sgn-eq[symmetric]*, *clarsimp*,  
*rule-tac*  $x=sgn$  - **in** *norm-matrix-bound*, *simp add: norm-sgn*) **+** *force*

**lemma** *op-norm-set-proptys*:

**fixes**  $A :: ('a::real\text{-}normed\text{-}algebra\text{-}1) \wedge 'n \wedge 'm$   
**shows** *bounded*  $\{\|A * v x\| \mid x. \|x\| = 1\}$   
**and** *bdd-above*  $\{\|A * v x\| \mid x. \|x\| = 1\}$   
**and**  $\{\|A * v x\| \mid x. \|x\| = 1\} \neq \{\}$   
**unfolding** *bounded-def bdd-above-def* **apply** *safe*  
**apply**(*rule-tac*  $x=0$  **in**  $exI$ , *rule-tac*  $x=\|(\chi \ i \ j. \|A \$ i \$ j\|) * v \ 1\|$  **in**  $exI$ )  
**apply**(*force simp: norm-matrix-bound dist-real-def*)  
**apply**(*rule-tac*  $x=\|(\chi \ i \ j. \|A \$ i \$ j\|) * v \ 1\|$  **in**  $exI$ , *force simp: norm-matrix-bound*)  
**using** *ex-norm-eq-1* **by** *blast*

**lemma** *op-norm-def*:  $\|A\|_{op} = Sup \ \{\|A * v x\| \mid x. \|x\| = 1\}$

**apply**(*rule antisym[OF onorm-le cSup-least[OF op-norm-set-proptys(3)]]*)  
**apply**(*case-tac*  $x = 0$ , *simp*)  
**apply**(*subst mult-norm-matrix-sgn-eq[symmetric]*, *simp*)  
**apply**(*rule cSup-upper[OF - op-norm-set-proptys(2)]*)  
**apply**(*force simp: norm-sgn*)  
**unfolding** *onorm-def*

**apply**(rule cSup-upper[OF - onorm-set-proptys(2)])  
**by** (simp add: image-def, clarsimp) (metis div-by-1)

**lemma** norm-matrix-le-op-norm:  $\|x\| = 1 \implies \|A * v x\| \leq \|A\|_{op}$   
**apply**(unfold onorm-def, rule cSup-upper[OF - onorm-set-proptys(2)])  
**unfolding** image-def **by** (clarsimp, rule-tac  $x=x$  in exI) simp

**lemma** op-norm-ge-0:  $0 \leq \|A\|_{op}$   
**using** ex-norm-eq-1 norm-ge-zero norm-matrix-le-op-norm basic-trans-rules(23) **by** blast

**lemma** norm-sgn-le-op-norm:  $\|A * v \text{sgn } x\| \leq \|A\|_{op}$   
**by** (cases  $x=0$ , simp-all add: norm-sgn norm-matrix-le-op-norm op-norm-ge-0)

**lemma** norm-matrix-le-mult-op-norm:  $\|A * v x\| \leq (\|A\|_{op}) * (\|x\|)$

**proof**—

**have**  $\|A * v x\| = (\|A * v \text{sgn } x\|) * (\|x\|)$   
**by**(simp add: mult-norm-matrix-sgn-eq)  
**also have**  $\dots \leq (\|A\|_{op}) * (\|x\|)$   
**using** norm-sgn-le-op-norm[of A] **by** (simp add: mult-mono')  
**finally show** ?thesis **by** simp

**qed**

**lemma** blin-matrix-vector-mult: bounded-linear  $((*v) A)$  **for**  $A :: ('a::\text{real-normed-algebra-1})^{n \times m}$   
**by** (unfold-locales) (auto intro: norm-matrix-le-mult-op-norm simp:  
mult commute matrix-vector-right-distrib vector-scaleR-commute)

**lemma** op-norm-eq-0:  $(\|A\|_{op} = 0) = (A = 0)$  **for**  $A :: ('a::\text{real-normed-field})^{n \times m}$   
**unfolding** onorm-eq-0[OF blin-matrix-vector-mult] **using** matrix-axis-0[of 1 A] **by** fastforce

**lemma** op-norm0:  $\|(0::('a::\text{real-normed-field})^{n \times m})\|_{op} = 0$   
**using** op-norm-eq-0[of 0] **by** simp

**lemma** op-norm-triangle:  $\|A + B\|_{op} \leq (\|A\|_{op}) + (\|B\|_{op})$   
**using** onorm-triangle[OF blin-matrix-vector-mult[of A] blin-matrix-vector-mult[of B]]  
matrix-vector-mult-add-rdistrib[symmetric, of A - B] **by** simp

**lemma** op-norm-scaleR:  $\|c *_R A\|_{op} = |c| * (\|A\|_{op})$   
**unfolding** onorm-scaleR[OF blin-matrix-vector-mult, symmetric] scaleR-vector-assoc ..

**lemma** op-norm-matrix-matrix-mult-le:  $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$

**proof**(rule onorm-le)

**have**  $0 \leq (\|A\|_{op})$   
**by**(rule onorm-pos-le[OF blin-matrix-vector-mult])  
**fix**  $x$  **have**  $\|A ** B * v x\| = \|A * v (B * v x)\|$   
**by** (simp add: matrix-vector-mul-assoc)  
**also have**  $\dots \leq (\|A\|_{op}) * (\|B * v x\|)$   
**by** (simp add: norm-matrix-le-mult-op-norm[of - B \* v x])  
**also have**  $\dots \leq (\|A\|_{op}) * ((\|B\|_{op}) * (\|x\|))$   
**using** norm-matrix-le-mult-op-norm[of B x] (0 ≤ ( $\|A\|_{op}$ )) mult-left-mono **by** blast  
**finally show**  $\|A ** B * v x\| \leq (\|A\|_{op}) * (\|B\|_{op}) * (\|x\|)$   
**by** simp

**qed**

**lemma** norm-matrix-vec-mult-le-transpose:

$\|x\| = 1 \implies (\|A * v x\|) \leq \text{sqrt} (\|transpose A ** A\|_{op}) * (\|x\|)$  **for**  $A :: \text{real}^{n \times n}$

**proof**—

**assume**  $\|x\| = 1$   
**have**  $(\|A * v x\|)^2 = (A * v x) \cdot (A * v x)$   
**using** dot-square-norm[of (A \* v x)] **by** simp  
**also have**  $\dots = x \cdot (transpose A * v (A * v x))$

```

    using vec-mult-inner by blast
  also have ... ≤ (||x||) * (||transpose A *v (A *v x)||)
    using norm-cauchy-schwarz by blast
  also have ... ≤ (||transpose A ** A||op) * (||x||)2
    apply(subst matrix-vector-mul-assoc)
    using norm-matrix-le-mult-op-norm[of transpose A ** A x]
    by (simp add: ⟨||x|| = 1⟩)
  finally have ((||A *v x||)2 ≤ (||transpose A ** A||op) * (||x||)2)
    by linarith
  thus (||A *v x||) ≤ sqrt ((||transpose A ** A||op) * (||x||))
    by (simp add: ⟨||x|| = 1⟩ real-le-rsqrt)
qed

```

**lemma** *op-norm-le-sum-column*:  $\|A\|_{op} \leq (\sum_{i \in UNIV}. \|column\ i\ A\|)$  **for**  $A :: real^{n \times m}$   
**proof**(*unfold op-norm-def*, *rule cSup-least[OF op-norm-set-proptys(3)]*, *clarsimp*)

```

  fix x :: real^n assume x-def: ||x|| = 1
  hence x-hyp: ∧i. ||x $ i|| ≤ 1
    by (simp add: norm-bound-component-le-cart)
  have (||A *v x||) = ||(∑ i ∈ UNIV. x $ i *s column i A)||
    by(subst matrix-mult-sum[of A], simp)
  also have ... ≤ (∑ i ∈ UNIV. ||x $ i *s column i A||)
    by (simp add: sum-norm-le)
  also have ... = (∑ i ∈ UNIV. (||x $ i||) * (||column i A||))
    by (simp add: mult-norm-matrix-sgn-eq)
  also have ... ≤ (∑ i ∈ UNIV. ||column i A||)
    using x-hyp by (simp add: mult-left-le-one-le sum-mono)
  finally show ||A *v x|| ≤ (∑ i ∈ UNIV. ||column i A||) .

```

qed

**lemma** *op-norm-le-transpose*:  $\|A\|_{op} \leq \|\text{transpose } A\|_{op}$  **for**  $A :: real^{n \times n}$

**proof**–

```

  have obs: ∀x. ||x|| = 1 ⟶ (||A *v x||) ≤ sqrt ((||transpose A ** A||op) * (||x||))
    using norm-matrix-vec-mult-le-transpose by blast
  have (||A||op) ≤ sqrt ((||transpose A ** A||op))
    using obs apply(unfold op-norm-def)
    by (rule cSup-least[OF op-norm-set-proptys(3)]) clarsimp
  hence ((||A||op)2 ≤ (||transpose A ** A||op))
    using power-mono[of (||A||op) - 2] op-norm-ge-0
    by (metis not-le real-less-lsqrt)
  also have ... ≤ (||transpose A||op) * (||A||op)
    using op-norm-matrix-matrix-mult-le by blast
  finally have ((||A||op)2 ≤ (||transpose A||op) * (||A||op))
    by linarith
  thus (||A||op) ≤ (||transpose A||op)
    using sq-le-cancel[of (||A||op)] op-norm-ge-0 by metis

```

qed

### 0.12.2 Matrix maximum norm

**abbreviation** *max-norm* ::  $real^{n \times m} \Rightarrow real$   $((1 - \|\cdot\|_{max})$  [65] 61)  
 where  $\|A\|_{max} \equiv Max\ (abs\ \cdot\ (entries\ A))$

**lemma** *max-norm-def*:  $\|A\|_{max} = Max\ \{|A\ \$\ i\ \$\ j| \mid i.j. i \in UNIV \wedge j \in UNIV\}$   
 by (*simp add: image-def*, *rule arg-cong[of - - Max]*, *blast*)

**lemma** *max-norm-set-proptys*: *finite*  $\{|A\ \$\ i\ \$\ j| \mid i.j. i \in UNIV \wedge j \in UNIV\}$  (**is finite** ?X)

**proof**–

```

  have ∧i. finite {|A $ i $ j| | j. j ∈ UNIV}
    using finite-Atleast-Atmost-nat by fastforce
  hence finite (⋃ i ∈ UNIV. {|A $ i $ j| | j. j ∈ UNIV}) (is finite ?Y)

```

using *finite-class.finite-UNIV* by *blast*  
 also have  $?X \subseteq ?Y$   
 by *auto*  
 ultimately show *?thesis*  
 using *finite-subset* by *blast*  
 qed

lemma *max-norm-ge-0*:  $0 \leq \|A\|_{\max}$   
 unfolding *max-norm-def*  
 apply (rule *order.trans* [OF *abs-ge-zero* [of  $A$  \$ - \$ -] *Max-ge*])  
 using *max-norm-set-proptys* by *auto*

lemma *op-norm-le-max-norm*:  
 fixes  $A :: \text{real}^{('n::\text{finite})} \times ('m::\text{finite})$   
 shows  $\|A\|_{\text{op}} \leq \text{real CARD}('m) * \text{real CARD}('n) * (\|A\|_{\max})$   
 apply (rule *onorm-le-matrix-component*)  
 unfolding *max-norm-def* by (rule *Max-ge* [OF *max-norm-set-proptys*]) *force*

lemma *sqrt-Sup-power2-eq-Sup-abs*:  
 $\text{finite } A \implies A \neq \{\} \implies \text{sqrt } (\text{Sup } \{(f\ i)^2 \mid i. i \in A\}) = \text{Sup } \{|f\ i| \mid i. i \in A\}$   
 proof (rule *sym*)  
 assume *assms*:  $\text{finite } A \ A \neq \{\}$   
 then obtain  $i$  where *i-def*:  $i \in A \wedge \text{Sup } \{(f\ i)^2 \mid i. i \in A\} = (f\ i)^2$   
 using *cSup-finite-ex* [of  $\{(f\ i)^2 \mid i. i \in A\}$ ] by *auto*  
 hence *lhs*:  $\text{sqrt } (\text{Sup } \{(f\ i)^2 \mid i. i \in A\}) = |f\ i|$   
 by *simp*  
 have  $\text{finite } \{(f\ i)^2 \mid i. i \in A\}$   
 using *assms* by *simp*  
 hence  $\forall j \in A. (f\ j)^2 \leq (f\ i)^2$   
 using *i-def* *cSup-upper* [of  $\{(f\ i)^2 \mid i. i \in A\}$ ] by *force*  
 hence  $\forall j \in A. |f\ j| \leq |f\ i|$   
 using *abs-le-square-iff* by *blast*  
 also have  $|f\ i| \in \{|f\ i| \mid i. i \in A\}$   
 using *i-def* by *auto*  
 ultimately show  $\text{Sup } \{|f\ i| \mid i. i \in A\} = \text{sqrt } (\text{Sup } \{(f\ i)^2 \mid i. i \in A\})$   
 using *cSup-mem-eq* [of  $|f\ i| \ \{|f\ i| \mid i. i \in A\}$ ] *lhs* by *auto*  
 qed

lemma *sqrt-Max-power2-eq-max-abs*:  
 $\text{finite } A \implies A \neq \{\} \implies \text{sqrt } (\text{Max } \{(f\ i)^2 \mid i. i \in A\}) = \text{Max } \{|f\ i| \mid i. i \in A\}$   
 apply (subst *cSup-eq-Max* [symmetric], *simp-all*) +  
 using *sqrt-Sup-power2-eq-Sup-abs* .

lemma *op-norm-diag-mat-eq*:  $\|\text{diag-mat } f\|_{\text{op}} = \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\}$  (is - = *Max ?A*)  
 proof (unfold *op-norm-def*)  
 have *obs*:  $\bigwedge x\ i. (f\ i)^2 * (x\ \$\ i)^2 \leq \text{Max } \{(f\ i)^2 \mid i. i \in \text{UNIV}\} * (x\ \$\ i)^2$   
 apply (rule *mult-right-mono* [OF *zero-le-power2*])  
 using *le-max-image-of-finite* [of  $\lambda i. (f\ i)^2$ ] by *simp*  
 {fix  $r$  assume  $r \in \{\|\text{diag-mat } f * v\ x\| \mid x. \|x\| = 1\}$   
 then obtain  $x$  where *x-def*:  $\|\text{diag-mat } f * v\ x\| = r \wedge \|x\| = 1$   
 by *blast*  
 hence  $r^2 = (\sum_{i \in \text{UNIV}} (f\ i)^2 * (x\ \$\ i)^2)$   
 unfolding *norm-vec-def* *L2-set-def* *matrix-vector-mul-diag-mat*  
 apply (*simp add: power-mult-distrib*)  
 by (*metis* (*no-types*, *lifting*) *x-def* *norm-ge-zero* *real-sqrt-ge-0-iff* *real-sqrt-pow2*)  
 also have  $\dots \leq (\text{Max } \{(f\ i)^2 \mid i. i \in \text{UNIV}\}) * (\sum_{i \in \text{UNIV}} (x\ \$\ i)^2)$   
 using *obs* [of  $x$ ] by (*simp add: sum-mono sum-distrib-left*)  
 also have  $\dots = \text{Max } \{(f\ i)^2 \mid i. i \in \text{UNIV}\}$   
 using *x-def* by (*simp add: norm-vec-def* *L2-set-def*)  
 finally have  $r \leq \text{sqrt } (\text{Max } \{(f\ i)^2 \mid i. i \in \text{UNIV}\})$

```

    using x-def real-le-rsqr by blast
  hence  $r \leq \text{Max } ?A$ 
    by (subst (asm) sqrt-Max-power2-eq-max-abs[of UNIV f], simp-all)
  hence 1:  $\forall x \in \{\| \text{diag-mat } f * v \, x \| \mid x. \|x\| = 1\}. x \leq \text{Max } ?A$ 
    unfolding diag-mat-def by blast
  obtain i where i-def:  $\text{Max } ?A = \| \text{diag-mat } f * v \, i \|$ 
    using cMax-finite-ex[of ?A] by force
  hence 2:  $\exists x \in \{\| \text{diag-mat } f * v \, x \| \mid x. \|x\| = 1\}. \text{Max } ?A \leq x$ 
    by (metis (mono-tags, lifting) abs-1 mem-Collect-eq norm-axis-eq order-refl real-norm-def)
  show  $\text{Sup } \{\| \text{diag-mat } f * v \, x \| \mid x. \|x\| = 1\} = \text{Max } ?A$ 
    by (rule cSup-eq[OF 1 2])
qed

```

lemma op-max-norms-eq-at-diag:  $\| \text{diag-mat } f \|_{op} = \| \text{diag-mat } f \|_{max}$

proof(rule antisym)

```

  have  $\{|f \, i| \mid i. i \in \text{UNIV}\} \subseteq \{| \text{diag-mat } f \, i \, j | \mid i \, j. i \in \text{UNIV} \wedge j \in \text{UNIV}\}$ 
    by (smt Collect-mono diag-mat-vec-nth-simps(1))
  thus  $\| \text{diag-mat } f \|_{op} \leq \| \text{diag-mat } f \|_{max}$ 
    unfolding op-norm-diag-mat-eq max-norm-def
    by (rule Max.subset-imp) (blast, simp only: finite-image-of-finite2)

```

next

```

  have  $\text{Sup } \{| \text{diag-mat } f \, i \, j | \mid i \, j. i \in \text{UNIV} \wedge j \in \text{UNIV}\} \leq \text{Sup } \{|f \, i| \mid i. i \in \text{UNIV}\}$ 
    apply(rule cSup-least, blast, clarify, case-tac i = j, simp)
    by (rule cSup-upper, blast, simp-all) (rule cSup-upper2, auto)
  thus  $\| \text{diag-mat } f \|_{max} \leq \| \text{diag-mat } f \|_{op}$ 
    unfolding op-norm-diag-mat-eq max-norm-def
    apply (subst cSup-eq-Max[symmetric], simp only: finite-image-of-finite2, blast)
    by (subst cSup-eq-Max[symmetric], simp, blast)

```

qed

end

## 0.13 Square Matrices

The general solution for affine systems of ODEs involves the exponential function. Unfortunately, this operation is only available in Isabelle for the type class “banach”. Hence, we define a type of square matrices and prove that it is an instance of this class.

theory *SQ-MTX*

imports *MTX-Norms*

begin

### 0.13.1 Definition

```

typedef 'm sq-mtx = UNIV::( $\text{real}^{'m} \times 'm$ ) set
  morphisms to-vec to-mtx by simp

```

```

declare to-mtx-inverse [simp]
  and to-vec-inverse [simp]

```

setup-lifting type-definition-sq-mtx

lift-definition sq-mtx-ith ::  $'m \text{ sq-mtx} \Rightarrow 'm \Rightarrow (\text{real}^{'m})$  (infixl  $\$ \$$  90) is (\$) .

lift-definition sq-mtx-vec-mult ::  $'m \text{ sq-mtx} \Rightarrow (\text{real}^{'m}) \Rightarrow (\text{real}^{'m})$  (infixl  $*_V$  90) is (\*v) .

lift-definition vec-sq-mtx-prod ::  $(\text{real}^{'m}) \Rightarrow 'm \text{ sq-mtx} \Rightarrow (\text{real}^{'m})$  is (v\*) .



**lift-definition**  $sq\text{-mtx}\text{-diag} :: (('m::finite) \Rightarrow real) \Rightarrow ('m::finite) sq\text{-mtx} \text{ (binder diag } 10) \text{ is diag-mat .}$

**lift-definition**  $sq\text{-mtx}\text{-transpose} :: ('m::finite) sq\text{-mtx} \Rightarrow 'm sq\text{-mtx} (-^\dagger) \text{ is transpose .}$

**lift-definition**  $sq\text{-mtx}\text{-inv} :: ('m::finite) sq\text{-mtx} \Rightarrow 'm sq\text{-mtx} (-^{-1} [90]) \text{ is matrix-inv .}$

**lift-definition**  $sq\text{-mtx}\text{-row} :: 'm \Rightarrow ('m::finite) sq\text{-mtx} \Rightarrow real^{'m} \text{ (row) is row .}$

**lift-definition**  $sq\text{-mtx}\text{-col} :: 'm \Rightarrow ('m::finite) sq\text{-mtx} \Rightarrow real^{'m} \text{ (col) is column .}$

**lemma**  $to\text{-vec}\text{-eq}\text{-ith}$ :  $(to\text{-vec } A) \$ i = A \$\$ i$   
by *transfer simp*

**lemma**  $to\text{-mtx}\text{-ith}[simp]$ :  
 $(to\text{-mtx } A) \$\$ i1 = A \$ i1$   
 $(to\text{-mtx } A) \$\$ i1 \$ i2 = A \$ i1 \$ i2$   
 by *(transfer, simp)+*

**lemma**  $to\text{-mtx}\text{-vec}\text{-lambda}\text{-ith}[simp]$ :  $to\text{-mtx } (\chi i j. x i j) \$\$ i1 \$ i2 = x i1 i2$   
by *(simp add: sq-mtx-ith-def)*

**lemma**  $sq\text{-mtx}\text{-eq}\text{-iff}$ :  
 shows  $A = B = (\forall i j. A \$\$ i \$ j = B \$\$ i \$ j)$   
 and  $A = B = (\forall i. A \$\$ i = B \$\$ i)$   
 by *(transfer, simp add: vec-eq-iff)+*

**lemma**  $sq\text{-mtx}\text{-diag}\text{-simps}[simp]$ :  
 $i = j \implies sq\text{-mtx}\text{-diag } f \$\$ i \$ j = f i$   
 $i \neq j \implies sq\text{-mtx}\text{-diag } f \$\$ i \$ j = 0$   
 $sq\text{-mtx}\text{-diag } f \$\$ i = axis i (f i)$   
 unfolding  $sq\text{-mtx}\text{-diag}\text{-def}$  by *(simp-all add: axis-def vec-eq-iff)*

**lemma**  $sq\text{-mtx}\text{-diag}\text{-vec}\text{-mult}$ :  $(diag i. f i) *_V s = (\chi i. f i * s \$ i)$   
by *(simp add: matrix-vector-mul-diag-mat sq-mtx-diag.abs-eq sq-mtx-vec-mult.abs-eq)*

**lemma**  $sq\text{-mtx}\text{-vec}\text{-mult}\text{-diag}\text{-axis}$ :  $(diag i. f i) *_V (axis i k) = axis i (f i * k)$   
unfolding  $sq\text{-mtx}\text{-diag}\text{-vec}\text{-mult}$   $axis\text{-def}$  by *auto*

**lemma**  $sq\text{-mtx}\text{-vec}\text{-mult}\text{-eq}$ :  $m *_V x = (\chi i. sum (\lambda j. (m \$\$ i \$ j) * (x \$ j))) UNIV$   
by *(transfer, simp add: matrix-vector-mult-def)*

**lemma**  $sq\text{-mtx}\text{-transpose}\text{-transpose}[simp]$ :  $(A^\dagger)^\dagger = A$   
by *(transfer, simp)*

**lemma**  $transpose\text{-mult}\text{-vec}\text{-canon}\text{-row}[simp]$ :  $(A^\dagger) *_V (e i) = row i A$   
by *transfer (simp add: row-def transpose-def axis-def matrix-vector-mult-def)*

**lemma**  $row\text{-ith}[simp]$ :  $row i A = A \$\$ i$   
by *transfer (simp add: row-def)*

**lemma**  $mtx\text{-vec}\text{-mult}\text{-canon}$ :  $A *_V (e i) = col i A$   
by *(transfer, simp add: matrix-vector-mult-basis)*

### 0.13.2 Ring of square matrices

**instantiation**  $sq\text{-mtx} :: (finite) ring$   
begin

**lift-definition** *plus-sq-mtx* :: 'a sq-mtx  $\Rightarrow$  'a sq-mtx  $\Rightarrow$  'a sq-mtx **is** (+) .

**lift-definition** *zero-sq-mtx* :: 'a sq-mtx **is** 0 .

**lift-definition** *uminus-sq-mtx* :: 'a sq-mtx  $\Rightarrow$  'a sq-mtx **is** uminus .

**lift-definition** *minus-sq-mtx* :: 'a sq-mtx  $\Rightarrow$  'a sq-mtx  $\Rightarrow$  'a sq-mtx **is** (-) .

**lift-definition** *times-sq-mtx* :: 'a sq-mtx  $\Rightarrow$  'a sq-mtx  $\Rightarrow$  'a sq-mtx **is** (\*\*) .

**declare** *plus-sq-mtx.rep-eq* [simp]  
**and** *minus-sq-mtx.rep-eq* [simp]

**instance** **apply** *intro-classes*

**by**(*transfer*, *simp add: algebra-simps matrix-mul-assoc matrix-add-rdistrib matrix-add-ldistrib*) +

**end**

**lemma** *sq-mtx-zero-ith*[simp]: 0 \$\$ i = 0  
**by** (*transfer*, *simp*)

**lemma** *sq-mtx-zero-nth*[simp]: 0 \$\$ i \$ j = 0  
**by** *transfer simp*

**lemma** *sq-mtx-plus-eq*: A + B = to-mtx ( $\chi$  i j. A \$\$ i \$ j + B \$\$ i \$ j)  
**by** *transfer (simp add: vec-eq-iff)*

**lemma** *sq-mtx-plus-ith*[simp]: (A + B) \$\$ i = A \$\$ i + B \$\$ i  
**unfolding** *sq-mtx-plus-eq* **by** (*simp add: vec-eq-iff*)

**lemma** *sq-mtx-uminus-eq*: - A = to-mtx ( $\chi$  i j. - A \$\$ i \$ j)  
**by** *transfer (simp add: vec-eq-iff)*

**lemma** *sq-mtx-minus-eq*: A - B = to-mtx ( $\chi$  i j. A \$\$ i \$ j - B \$\$ i \$ j)  
**by** *transfer (simp add: vec-eq-iff)*

**lemma** *sq-mtx-minus-ith*[simp]: (A - B) \$\$ i = A \$\$ i - B \$\$ i  
**unfolding** *sq-mtx-minus-eq* **by** (*simp add: vec-eq-iff*)

**lemma** *sq-mtx-times-eq*: A \* B = to-mtx ( $\chi$  i j. sum ( $\lambda k$ . A \$\$ i \$ k \* B \$\$ k \$ j) UNIV)  
**by** *transfer (simp add: matrix-matrix-mult-def)*

**lemma** *sq-mtx-plus-diag-diag*[simp]: *sq-mtx-diag* f + *sq-mtx-diag* g = (*diag* i. f i + g i)  
**by** (*subst sq-mtx-eq-iff*) (*simp add: axis-def*)

**lemma** *sq-mtx-minus-diag-diag*[simp]: *sq-mtx-diag* f - *sq-mtx-diag* g = (*diag* i. f i - g i)  
**by** (*subst sq-mtx-eq-iff*) (*simp add: axis-def*)

**lemma** *sum-sq-mtx-diag*[simp]: ( $\sum n < m$ . *sq-mtx-diag* (g n)) = (*diag* i.  $\sum n < m$ . (g n i)) **for** m::nat  
**by** (*induct m*, *simp*, *subst sq-mtx-eq-iff*, *simp-all*)

**lemma** *sq-mtx-mult-diag-diag*[simp]: *sq-mtx-diag* f \* *sq-mtx-diag* g = (*diag* i. f i \* g i)  
**by** (*simp add: matrix-mul-diag-diag sq-mtx-diag.abs-eq times-sq-mtx.abs-eq*)

**lemma** *sq-mtx-mult-diagl*: (*diag* i. f i) \* A = to-mtx ( $\chi$  i j. f i \* A \$\$ i \$ j)  
**by** *transfer (simp add: matrix-mul-diag-matl)*

**lemma** *sq-mtx-mult-diagr*: A \* (*diag* i. f i) = to-mtx ( $\chi$  i j. A \$\$ i \$ j \* f j)  
**by** *transfer (simp add: matrix-matrix-mult-diag-matr)*

**lemma** *mtx-vec-mult-0l*[simp]:  $0 *_V x = 0$   
**by** (*simp add: sq-mtx-vec-mult.abs-eq zero-sq-mtx-def*)

**lemma** *mtx-vec-mult-0r*[simp]:  $A *_V 0 = 0$   
**by** (*transfer, simp*)

**lemma** *mtx-vec-mult-add-rdistr*:  $(A + B) *_V x = A *_V x + B *_V x$   
**unfolding** *plus-sq-mtx-def*  
**apply** (*transfer*)  
**by** (*simp add: matrix-vector-mult-add-rdistrib*)

**lemma** *mtx-vec-mult-add-rdistl*:  $A *_V (x + y) = A *_V x + A *_V y$   
**unfolding** *plus-sq-mtx-def*  
**apply** *transfer*  
**by** (*simp add: matrix-vector-right-distrib*)

**lemma** *mtx-vec-mult-minus-rdistrib*:  $(A - B) *_V x = A *_V x - B *_V x$   
**unfolding** *minus-sq-mtx-def* **by** (*transfer, simp add: matrix-vector-mult-diff-rdistrib*)

**lemma** *mtx-vec-mult-minus-ldistrib*:  $A *_V (x - y) = A *_V x - A *_V y$   
**by** (*metis (no-types, lifting) add-diff-cancel diff-add-cancel*  
*matrix-vector-right-distrib sq-mtx-vec-mult.rep-eq*)

**lemma** *sq-mtx-times-vec-assoc*:  $(A * B) *_V x = A *_V (B *_V x)$   
**by** (*transfer, simp add: matrix-vector-mul-assoc*)

**lemma** *sq-mtx-vec-mult-sum-cols*:  $A *_V x = \text{sum } (\lambda i. x \$ i *_R \text{col } i A) \text{ UNIV}$   
**by** (*transfer*) (*simp add: matrix-mult-sum scalar-mult-eq-scaleR*)

### 0.13.3 Real normed vector space of square matrices

**instantiation** *sq-mtx* :: (*finite*) *real-normed-vector*  
**begin**

**definition** *norm-sq-mtx* :: '*a* *sq-mtx*  $\Rightarrow$  *real* **where**  $\|A\| = \|to\text{-vec } A\|_{op}$

**lift-definition** *scaleR-sq-mtx* :: *real*  $\Rightarrow$  '*a* *sq-mtx*  $\Rightarrow$  '*a* *sq-mtx* **is** *scaleR* .

**definition** *sgn-sq-mtx* :: '*a* *sq-mtx*  $\Rightarrow$  '*a* *sq-mtx*  
**where** *sgn-sq-mtx* *A* = (*inverse* ( $\|A\|$ ))  $*_R A$

**definition** *dist-sq-mtx* :: '*a* *sq-mtx*  $\Rightarrow$  '*a* *sq-mtx*  $\Rightarrow$  *real*  
**where** *dist-sq-mtx* *A B* =  $\|A - B\|$

**definition** *uniformity-sq-mtx* :: ('*a* *sq-mtx*  $\times$  '*a* *sq-mtx*) *filter*  
**where** *uniformity-sq-mtx* = (*INF*  $e \in \{0 < ..\}$ . *principal*  $\{(x, y). \text{dist } x \ y < e\}$ )

**definition** *open-sq-mtx* :: '*a* *sq-mtx* *set*  $\Rightarrow$  *bool*  
**where** *open-sq-mtx* *U* =  $(\forall x \in U. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U)$

**instance** **apply** *intro-classes*  
**unfolding** *sgn-sq-mtx-def open-sq-mtx-def dist-sq-mtx-def uniformity-sq-mtx-def*  
**prefer** 10  
**apply** (*transfer, simp add: norm-sq-mtx-def op-norm-triangle*)  
**prefer** 9  
**apply** (*simp-all add: norm-sq-mtx-def zero-sq-mtx-def op-norm-eq-0*)  
**by** (*transfer, simp add: norm-sq-mtx-def op-norm-scaleR algebra-simps*) +

**end**

**lemma** *sq-mtx-scaleR-eq*:  $c *_R A = \text{to-mtx } (\chi \ i \ j. c *_R A \ \$\$ \ i \ \$ \ j)$   
**by** *transfer (simp add: vec-eq-iff)*

**lemma** *scaleR-to-mtx-ith[simp]*:  $c *_R (\text{to-mtx } A) \ \$\$ \ i1 \ \$ \ i2 = c *_R A \ \$ \ i1 \ \$ \ i2$   
**by** *transfer (simp add: scaleR-vec-def)*

**lemma** *sq-mtx-scaleR-ith[simp]*:  $(c *_R A) \ \$\$ \ i = (c *_R (A \ \$\$ \ i))$   
**by** *(unfold scaleR-sq-mtx-def, transfer, simp)*

**lemma** *scaleR-sq-mtx-diag*:  $c *_R \text{sq-mtx-diag } f = (\text{diag } i. c *_R f \ i)$   
**by** *(subst sq-mtx-eq-iff, simp add: axis-def)*

**lemma** *scaleR-mtx-vec-assoc*:  $(c *_R A) *_V x = c *_R (A *_V x)$   
**unfolding** *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply** *simp*  
**by** *(simp add: scaleR-matrix-vector-assoc)*

**lemma** *mtx-vec-scaleR-commute*:  $A *_V (c *_R x) = c *_R (A *_V x)$   
**unfolding** *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply** *(simp, transfer)*  
**by** *(simp add: vector-scaleR-commute)*

**lemma** *mtx-times-scaleR-commute*:  $A *_R (c *_R B) = c *_R (A *_R B)$  **for**  $A :: ('n :: \text{finite}) \text{sq-mtx}$   
**unfolding** *sq-mtx-scaleR-eq sq-mtx-times-eq*  
**apply** *(simp add: to-mtx-inject)*  
**apply** *(simp add: vec-eq-iff fun-eq-iff)*  
**by** *(simp add: semiring-normalization-rules(19) vector-space-over-itself.scale-sum-right)*

**lemma** *le-mtx-norm*:  $m \in \{\|A *_V x\| \mid x. \|x\| = 1\} \implies m \leq \|A\|$   
**using** *cSup-upper[of - {\| (to-vec A) \*\_V x \| \mid x. \|x\| = 1}]*  
**by** *(simp add: op-norm-set-proptys(2) op-norm-def norm-sq-mtx-def sq-mtx-vec-mult.rep-eq)*

**lemma** *norm-vec-mult-le*:  $\|A *_V x\| \leq (\|A\|) * (\|x\|)$   
**by** *(simp add: norm-matrix-le-mult-op-norm norm-sq-mtx-def sq-mtx-vec-mult.rep-eq)*

**lemma** *bounded-bilinear-sq-mtx-vec-mult*: *bounded-bilinear*  $(\lambda A \ s. A *_V s)$   
**apply** *(rule bounded-bilinear.intro, simp-all add: mtx-vec-mult-add-rdistr  
mtx-vec-mult-add-rdistl scaleR-mtx-vec-assoc mtx-vec-scaleR-commute)*  
**by** *(rule-tac x=1 in exI, auto intro!: norm-vec-mult-le)*

**lemma** *norm-sq-mtx-def2*:  $\|A\| = \text{Sup } \{\|A *_V x\| \mid x. \|x\| = 1\}$   
**unfolding** *norm-sq-mtx-def op-norm-def sq-mtx-vec-mult-def* **by** *simp*

**lemma** *norm-sq-mtx-def3*:  $\|A\| = (\text{SUP } x. (\|A *_V x\|) / (\|x\|))$   
**unfolding** *norm-sq-mtx-def onorm-def sq-mtx-vec-mult-def* **by** *simp*

**lemma** *norm-sq-mtx-diag*:  $\|\text{sq-mtx-diag } f\| = \text{Max } \{|f \ i| \mid i. i \in \text{UNIV}\}$   
**unfolding** *norm-sq-mtx-def* **apply** *transfer*  
**by** *(rule op-norm-diag-mat-eq)*

**lemma** *sq-mtx-norm-le-sum-col*:  $\|A\| \leq (\sum i \in \text{UNIV}. \|\text{col } i \ A\|)$   
**using** *op-norm-le-sum-column[of to-vec A]*  
**apply** *(simp add: norm-sq-mtx-def)*  
**by** *(transfer, simp add: op-norm-le-sum-column)*

**lemma** *norm-le-transpose*:  $\|A\| \leq \|A^\dagger\|$   
**unfolding** *norm-sq-mtx-def* **by** *transfer (rule op-norm-le-transpose)*

**lemma** *norm-eq-norm-transpose[simp]*:  $\|A^\dagger\| = \|A\|$   
**using** *norm-le-transpose[of A] and norm-le-transpose[of A^\dagger]* **by** *simp*

**lemma** *norm-column-le-norm*:  $\|A \ \$\$ \ i\| \leq \|A\|$

**using** *norm-vec-mult-le*[*of*  $A^\dagger \in i$ ] **by** *simp*

### 0.13.4 Real normed algebra of square matrices

**instantiation** *sq-mtx* :: (*finite*) *real-normed-algebra-1*  
**begin**

**lift-definition** *one-sq-mtx* :: '*a sq-mtx* **is** *to-mtx* (*mat* 1) .

**lemma** *sq-mtx-one-idty*:  $1 * A = A * 1 = A$  **for**  $A :: 'a \text{ sq-mtx}$   
**by**(*transfer*, *transfer*, *unfold mat-def matrix-matrix-mult-def*, *simp add: vec-eq-iff*)+

**lemma** *sq-mtx-norm-1*:  $\|(1 :: 'a \text{ sq-mtx})\| = 1$   
**unfolding** *one-sq-mtx-def norm-sq-mtx-def*  
**apply**(*simp add: op-norm-def*)  
**apply**(*subst cSup-eq[of - 1]*)  
**using** *ex-norm-eq-1* **by** *auto*

**lemma** *sq-mtx-norm-times*:  $\|A * B\| \leq (\|A\|) * (\|B\|)$  **for**  $A :: 'a \text{ sq-mtx}$   
**unfolding** *norm-sq-mtx-def times-sq-mtx-def* **by**(*simp add: op-norm-matrix-matrix-mult-le*)

**instance**  
**apply** *intro-classes*  
**apply**(*simp-all add: sq-mtx-one-idty sq-mtx-norm-1 sq-mtx-norm-times*)  
**apply**(*simp-all add: to-mtx-inject vec-eq-iff one-sq-mtx-def zero-sq-mtx-def mat-def*)  
**by**(*transfer*, *simp add: scalar-matrix-assoc matrix-scalar-ac*)+

**end**

**lemma** *sq-mtx-one-ith-simps*[*simp*]:  $1 \$\$ i \$ i = 1 \ i \neq j \implies 1 \$\$ i \$ j = 0$   
**unfolding** *one-sq-mtx-def mat-def* **by** *simp-all*

**lemma** *of-nat-eq-sq-mtx-diag*[*simp*]: *of-nat*  $m = (\text{diag } i. m)$   
**by** (*induct*  $m$ ) (*simp*, *subst sq-mtx-eq-iff*, *simp add: axis-def*)+

**lemma** *mtx-vec-mult-1*[*simp*]:  $1 *_V s = s$   
**by** (*auto simp: sq-mtx-vec-mult-def one-sq-mtx-def*  
*mat-def vec-eq-iff matrix-vector-mult-def*)

**lemma** *sq-mtx-diag-one*[*simp*]:  $(\text{diag } i. 1) = 1$   
**by** (*subst sq-mtx-eq-iff*, *simp add: one-sq-mtx-def mat-def axis-def*)

**abbreviation** *mtx-invertible*  $A \equiv \text{invertible } (\text{to-vec } A)$

**lemma** *mtx-invertible-def*:  $\text{mtx-invertible } A \longleftrightarrow (\exists A'. A' * A = 1 \wedge A * A' = 1)$   
**apply** (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def invertible-def*, *clarsimp*, *safe*)  
**apply**(*rule-tac x=to-mtx A' in exI*, *simp*)  
**by** (*rule-tac x=to-vec A' in exI*, *simp add: to-mtx-inject*)

**lemma** *mtx-invertibleI*:  
**assumes**  $A * B = 1$  **and**  $B * A = 1$   
**shows** *mtx-invertible*  $A$   
**using** *assms unfolding mtx-invertible-def* **by** *auto*

**lemma** *mtx-invertibleD*[*simp*]:  
**assumes** *mtx-invertible*  $A$   
**shows**  $A^{-1} * A = 1$  **and**  $A * A^{-1} = 1$   
**apply** (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def*)  
**using** *assms* **by** *simp-all*

**lemma** *mtx-invertible-inv*[simp]: *mtx-invertible*  $A \implies \text{mtx-invertible } (A^{-1})$   
**using** *mtx-invertibleD* *mtx-invertibleI* **by** *blast*

**lemma** *mtx-invertible-one*[simp]: *mtx-invertible* 1  
**by** (*simp add: one-sq-mtx.rep-eq*)

**lemma** *sq-mtx-inv-unique*:  
**assumes**  $A * B = 1$  **and**  $B * A = 1$   
**shows**  $A^{-1} = B$   
**by** (*metis (no-types, lifting) assms mtx-invertibleD(2) mtx-invertibleI mult.assoc sq-mtx-one-idty(1)*)

**lemma** *sq-mtx-inv-idempotent*[simp]: *mtx-invertible*  $A \implies A^{-1-1} = A$   
**using** *mtx-invertibleD* *sq-mtx-inv-unique* **by** *blast*

**lemma** *sq-mtx-inv-mult*:  
**assumes** *mtx-invertible*  $A$  **and** *mtx-invertible*  $B$   
**shows**  $(A * B)^{-1} = B^{-1} * A^{-1}$   
**by** (*simp add: assms matrix-inv-matrix-mul sq-mtx-inv-def times-sq-mtx-def*)

**lemma** *sq-mtx-inv-one*[simp]:  $1^{-1} = 1$   
**by** (*simp add: sq-mtx-inv-unique*)

**definition** *similar-sq-mtx* ::  $(n::\text{finite}) \text{ sq-mtx} \Rightarrow 'n \text{ sq-mtx} \Rightarrow \text{bool}$  (**infixr**  $\sim$  25)  
**where**  $(A \sim B) \longleftrightarrow (\exists P. \text{mtx-invertible } P \wedge A = P^{-1} * B * P)$

**lemma** *similar-sq-mtx-matrix*:  $(A \sim B) = \text{similar-matrix } (\text{to-vec } A) (\text{to-vec } B)$   
**apply**(*unfold similar-matrix-def similar-sq-mtx-def, safe*)  
**apply** (*metis sq-mtx-inv.rep-eq times-sq-mtx.rep-eq*)  
**by** (*metis UNIV-I sq-mtx-inv.abs-eq times-sq-mtx.abs-eq to-mtx-inverse to-vec-inverse*)

**lemma** *similar-sq-mtx-refl*[simp]:  $A \sim A$   
**by** (*unfold similar-sq-mtx-def, rule-tac x=1 in exI, simp*)

**lemma** *similar-sq-mtx-simm*:  $A \sim B \implies B \sim A$   
**apply**(*unfold similar-sq-mtx-def, clarsimp*)  
**apply**(*rule-tac x=P^{-1} in exI, simp add: mult.assoc*)  
**by** (*metis mtx-invertibleD(2) mult.assoc mult.left-neutral*)

**lemma** *similar-sq-mtx-trans*:  $A \sim B \implies B \sim C \implies A \sim C$   
**unfolding** *similar-sq-mtx-matrix* **using** *similar-matrix-trans* **by** *blast*

**lemma** *power-sq-mtx-diag*:  $(\text{sq-mtx-diag } f)^{\wedge n} = (\text{diag } i. f i^{\wedge n})$   
**by** (*induct n, simp-all*)

**lemma** *power-similar-sq-mtx-diag-eq*:  
**assumes** *mtx-invertible*  $P$   
**and**  $A = P^{-1} * (\text{sq-mtx-diag } f) * P$   
**shows**  $A^{\wedge n} = P^{-1} * (\text{diag } i. f i^{\wedge n}) * P$   
**proof**(*induct n, simp-all add: assms*)  
**fix**  $n::\text{nat}$   
**have**  $P^{-1} * \text{sq-mtx-diag } f * P * (P^{-1} * (\text{diag } i. f i^{\wedge n}) * P) =$   
 $P^{-1} * \text{sq-mtx-diag } f * (\text{diag } i. f i^{\wedge n}) * P$   
**by** (*metis (no-types, lifting) assms(1) mtx-invertibleD(2) mult.assoc mult.right-neutral*)  
**also have**  $\dots = P^{-1} * (\text{diag } i. f i * f i^{\wedge n}) * P$   
**by** (*simp add: mult.assoc*)  
**finally show**  $P^{-1} * \text{sq-mtx-diag } f * P * (P^{-1} * (\text{diag } i. f i^{\wedge n}) * P) =$   
 $P^{-1} * (\text{diag } i. f i * f i^{\wedge n}) * P$ .  
**qed**

**lemma** *power-similar-sq-mtx-diag*:  
**assumes**  $A \sim (\text{sq-mtx-diag } f)$   
**shows**  $A^n \sim (\text{diag } i. f \ i^n)$   
**using** *assms power-similar-sq-mtx-diag-eq*  
**unfolding** *similar-sq-mtx-def* **by** *blast*

### 0.13.5 Banach space of square matrices

**lemma** *Cauchy-cols*:  
**fixes**  $X :: \text{nat} \Rightarrow ('a::\text{finite}) \text{ sq-mtx}$   
**assumes** *Cauchy*  $X$   
**shows** *Cauchy*  $(\lambda n. \text{col } i \ (X \ n))$   
**proof**(*unfold Cauchy-def dist-norm, clarsimp*)  
**fix**  $\varepsilon::\text{real}$  **assume**  $\varepsilon > 0$   
**then obtain**  $M$  **where**  $M\text{-def}:\forall m \geq M. \forall n \geq M. \|X \ m - X \ n\| < \varepsilon$   
**using**  $(\text{Cauchy } X)$  **unfolding** *Cauchy-def* **by**(*simp add: dist-sq-mtx-def*) *metis*  
**{fix**  $m \ n$  **assume**  $m \geq M$  **and**  $n \geq M$   
**hence**  $\varepsilon > \|X \ m - X \ n\|$   
**using**  $M\text{-def}$  **by** *blast*  
**moreover have**  $\|X \ m - X \ n\| \geq \|(X \ m - X \ n) *_V e \ i\|$   
**by**(*rule le-mtx-norm[of - X \ m - X \ n], force*)  
**moreover have**  $\|(X \ m - X \ n) *_V e \ i\| = \|X \ m *_V e \ i - X \ n *_V e \ i\|$   
**by** (*simp add: mtx-vec-mult-minus-rdistrib*)  
**moreover have**  $\dots = \|\text{col } i \ (X \ m) - \text{col } i \ (X \ n)\|$   
**by** (*simp add: mtx-vec-mult-minus-rdistrib mtx-vec-mult-canon*)  
**ultimately have**  $\|\text{col } i \ (X \ m) - \text{col } i \ (X \ n)\| < \varepsilon$   
**by** *linarith* }  
**thus**  $\exists M. \forall m \geq M. \forall n \geq M. \|\text{col } i \ (X \ m) - \text{col } i \ (X \ n)\| < \varepsilon$   
**by** *blast*  
**qed**

**lemma** *col-convergence*:  
**assumes**  $\forall i. (\lambda n. \text{col } i \ (X \ n)) \longrightarrow L \ \$ \ i$   
**shows**  $X \longrightarrow \text{to-mtx} \ (\text{transpose } L)$   
**proof**(*unfold LIMSEQ-def dist-norm, clarsimp*)  
**let**  $?L = \text{to-mtx} \ (\text{transpose } L)$   
**let**  $?a = \text{CARD}('a)$  **fix**  $\varepsilon::\text{real}$  **assume**  $\varepsilon > 0$   
**hence**  $\varepsilon / ?a > 0$  **by** *simp*  
**hence**  $\forall i. \exists N. \forall n \geq N. \|\text{col } i \ (X \ n) - L \ \$ \ i\| < \varepsilon / ?a$   
**using** *assms unfolding LIMSEQ-def dist-norm convergent-def* **by** *blast*  
**then obtain**  $N$  **where**  $\forall i. \forall n \geq N. \|\text{col } i \ (X \ n) - L \ \$ \ i\| < \varepsilon / ?a$   
**using** *finite-nat-minimal-witness[of \lambda i n. \|\text{col } i \ (X \ n) - L \ \\$ \ i\| < \varepsilon / ?a]* **by** *blast*  
**also have**  $\bigwedge i n. (\text{col } i \ (X \ n) - L \ \$ \ i) = (\text{col } i \ (X \ n - ?L))$   
**unfolding** *minus-sq-mtx-def* **by**(*transfer, simp add: transpose-def vec-eq-iff column-def*)  
**ultimately have**  $N\text{-def}:\forall i. \forall n \geq N. \|\text{col } i \ (X \ n - ?L)\| < \varepsilon / ?a$   
**by** *auto*  
**have**  $\forall n \geq N. \|X \ n - ?L\| < \varepsilon$   
**proof**(*rule allI, rule impI*)  
**fix**  $n::\text{nat}$  **assume**  $N \leq n$   
**hence**  $\forall i. \|\text{col } i \ (X \ n - ?L)\| < \varepsilon / ?a$   
**using**  $N\text{-def}$  **by** *blast*  
**hence**  $(\sum i \in UNIV. \|\text{col } i \ (X \ n - ?L)\|) < (\sum (i::'a) \in UNIV. \varepsilon / ?a)$   
**using** *sum-strict-mono[of - \lambda i. \|\text{col } i \ (X \ n - ?L)\|]* **by** *force*  
**moreover have**  $\|X \ n - ?L\| \leq (\sum i \in UNIV. \|\text{col } i \ (X \ n - ?L)\|)$   
**using** *sq-mtx-norm-le-sum-col* **by** *blast*  
**moreover have**  $(\sum (i::'a) \in UNIV. \varepsilon / ?a) = \varepsilon$   
**by** *force*  
**ultimately show**  $\|X \ n - ?L\| < \varepsilon$   
**by** *linarith*  
**qed**

thus  $\exists no. \forall n \geq no. \|X\ n - ?L\| < \varepsilon$   
 by *blast*  
 qed

instance *sq-mtx* :: (*finite*) *banach*

proof(*standard*)

fix *X* :: *nat*  $\Rightarrow$  'a *sq-mtx*

assume *Cauchy X*

hence  $\bigwedge i. \text{Cauchy } (\lambda n. \text{col } i \ (X\ n))$

using *Cauchy-cols* by *blast*

hence obs:  $\forall i. \exists ! L. (\lambda n. \text{col } i \ (X\ n)) \longrightarrow L$

using *Cauchy-convergent convergent-def LIMSEQ-unique* by *fastforce*

define *L* where *L* = ( $\chi\ i. \lim (\lambda n. \text{col } i \ (X\ n))$ )

hence  $\forall i. (\lambda n. \text{col } i \ (X\ n)) \longrightarrow L\ \$\ i$

using obs *theI-unique*[of  $\lambda L. (\lambda n. \text{col } - \ (X\ n)) \longrightarrow L\ L\ \$\ -$ ] by (*simp add: lim-def*)

thus *convergent X*

using *col-convergence unfolding convergent-def* by *blast*

qed

lemma *exp-similar-sq-mtx-diag-eq*:

assumes *mtx-invertible P*

and  $A = P^{-1} * (\text{diag } i. f\ i) * P$

shows  $\exp A = P^{-1} * \exp (\text{diag } i. f\ i) * P$

proof(*unfold exp-def power-similar-sq-mtx-diag-eq*[*OF assms*])

have  $(\sum n. P^{-1} * (\text{diag } i. f\ i \wedge n) * P /_R \text{fact } n) =$

$(\sum n. P^{-1} * ((\text{diag } i. f\ i \wedge n) /_R \text{fact } n) * P)$

by *simp*

also have  $\dots = (\sum n. P^{-1} * ((\text{diag } i. f\ i \wedge n) /_R \text{fact } n)) * P$

apply(*subst suminf-mult*[*OF bounded-linear.summable*[*OF bounded-linear-mult-right*]])

unfolding *power-sq-mtx-diag*[*symmetric*] by (*simp-all add: summable-exp-generic*)

also have  $\dots = P^{-1} * (\sum n. (\text{diag } i. f\ i \wedge n) /_R \text{fact } n) * P$

apply(*subst suminf-mult*[of  $- P^{-1}$ ])

unfolding *power-sq-mtx-diag*[*symmetric*]

by (*simp-all add: summable-exp-generic*)

finally show  $(\sum n. P^{-1} * (\text{diag } i. f\ i \wedge n) * P /_R \text{fact } n) =$

$P^{-1} * (\sum n. \text{sq-mtx-diag } f \wedge n /_R \text{fact } n) * P$

unfolding *power-sq-mtx-diag* by *simp*

qed

lemma *exp-similar-sq-mtx-diag*:

assumes  $A \sim \text{sq-mtx-diag } f$

shows  $\exp A \sim \exp (\text{sq-mtx-diag } f)$

using *assms exp-similar-sq-mtx-diag-eq*

unfolding *similar-sq-mtx-def* by *blast*

lemma *suminf-sq-mtx-diag*:

assumes  $\forall i. (\lambda n. f\ n\ i) \text{ sums } (\text{suminf } (\lambda n. f\ n\ i))$

shows  $(\sum n. (\text{diag } i. f\ n\ i)) = (\text{diag } i. \sum n. f\ n\ i)$

proof(*rule suminfI, unfold sums-def LIMSEQ-iff, clarsimp simp: norm-sq-mtx-diag*)

let  $?g = \lambda n\ i. |(\sum n < n. f\ n\ i) - (\sum n. f\ n\ i)|$

fix *r::real* assume  $r > 0$

have  $\forall i. \exists no. \forall n \geq no. ?g\ n\ i < r$

using *assms* ( $r > 0$ ) unfolding *sums-def LIMSEQ-iff* by *clarsimp*

then obtain *N* where key:  $\forall i. \forall n \geq N. ?g\ n\ i < r$

using *finite-nat-minimal-witness*[of  $\lambda i\ n. ?g\ n\ i < r$ ] by *blast*

{fix *n::nat*

assume  $n \geq N$

obtain *i* where *i-def*:  $\text{Max } \{x. \exists i. x = ?g\ n\ i\} = ?g\ n\ i$

using *cMax-finite-ex*[of  $\{x. \exists i. x = ?g\ n\ i\}$ ] by *auto*

hence  $?g\ n\ i < r$



```

    using key ⟨n ≥ N⟩ by blast
    hence Max {x. ∃ i. x = ?g n i} < r
    unfolding i-def[symmetric] .}
  thus ∃ N. ∀ n ≥ N. Max {x. ∃ i. x = ?g n i} < r
  by blast
qed

```

```

lemma exp-sq-mtx-diag: exp (sq-mtx-diag f) = (diag i. exp (f i))
  apply(unfold exp-def, simp add: power-sq-mtx-diag scaleR-sq-mtx-diag)
  apply(rule suminf-sq-mtx-diag)
  using exp-converges[of f -]
  unfolding sums-def LIMSEQ-iff exp-def by force

```

```

lemma exp-scaleR-diagonal1:
  assumes mtx-invertible P and A = P-1 * (diag i. f i) * P
  shows exp (t *R A) = P-1 * (diag i. exp (t * f i)) * P
proof-
  have exp (t *R A) = exp (P-1 * (t *R sq-mtx-diag f) * P)
  using assms by simp
  also have ... = P-1 * (diag i. exp (t * f i)) * P
  by (metis assms(1) exp-similar-sq-mtx-diag-eq exp-sq-mtx-diag scaleR-sq-mtx-diag)
  finally show exp (t *R A) = P-1 * (diag i. exp (t * f i)) * P .
qed

```

```

lemma exp-scaleR-diagonal2:
  assumes mtx-invertible P and A = P * (diag i. f i) * P-1
  shows exp (t *R A) = P * (diag i. exp (t * f i)) * P-1
  apply(subst sq-mtx-inv-idempotent[OF assms(1), symmetric])
  apply(rule exp-scaleR-diagonal1)
  by (simp-all add: assms)

```

### 0.13.6 Examples

```

definition mtx A = to-mtx (vector (map vector A))

```

```

lemma vector-nth-eq: (vector A) $ i = foldr (λx f n. (f (n + 1))(n := x)) A (λn x. 0) 1 i
  unfolding vector-def by simp

```

```

lemma mtx-ith-eq[simp]: mtx A $$ i $ j = foldr (λx f n. (f (n + 1))(n := x))
  (map (λl. vec-lambda (foldr (λx f n. (f (n + 1))(n := x)) l (λn x. 0) 1)) A) (λn x. 0) 1 i $ j
  unfolding mtx-def vector-def by (simp add: vector-nth-eq)

```

### 2x2 matrices

```

lemma mtx2-eq-iff: (mtx
  ([a1, b1] #
   [c1, d1] # []) :: 2 sq-mtx) = mtx
  ([a2, b2] #
   [c2, d2] # []) ⟷ a1 = a2 ∧ b1 = b2 ∧ c1 = c2 ∧ d1 = d2
  apply(simp add: sq-mtx-eq-iff, safe)
  using exhaust-2 by force+

```

```

lemma mtx2-to-mtx: mtx
  ([a, b] #
   [c, d] # []) =
  to-mtx (χ i j::2. if i=1 ∧ j=1 then a
    else (if i=1 ∧ j=2 then b
      else (if i=2 ∧ j=1 then c
        else d)))
  apply(subst sq-mtx-eq-iff)

```

**using** *exhaust-2* **by** *force*

**abbreviation** *diag2* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  2 *sq-mtx*

**where** *diag2*  $\iota_1$   $\iota_2 \equiv$  *mtx*

$([\iota_1, 0] \#$   
 $[0, \iota_2] \# [])$

**lemma** *diag2-eq*: *diag2* ( $\iota$  1) ( $\iota$  2) = (diag *i.*  $\iota$  *i*)

**apply**(*simp add: sq-mtx-eq-iff*)

**using** *exhaust-2* **by** (*force simp: axis-def*)

**lemma** *one-mtx2*: (1::2 *sq-mtx*) = *diag2* 1 1

**apply**(*subst sq-mtx-eq-iff*)

**using** *exhaust-2* **by** *force*

**lemma** *zero-mtx2*: (0::2 *sq-mtx*) = *diag2* 0 0

**by** (*simp add: sq-mtx-eq-iff*)

**lemma** *scaleR-mtx2*: *k* \*<sub>*R*</sub> *mtx*

$([a, b] \#$   
 $[c, d] \# []) =$  *mtx*

$([k*a, k*b] \#$   
 $[k*c, k*d] \# [])$

**by** (*simp add: sq-mtx-eq-iff*)

**lemma** *uminus-mtx2*:  $-$  *mtx*

$([a, b] \#$   
 $[c, d] \# []) =$  (*mtx*

$([-a, -b] \#$   
 $[-c, -d] \# [])::2$  *sq-mtx*)

**by** (*simp add: sq-mtx-uminus-eq sq-mtx-eq-iff*)

**lemma** *plus-mtx2*: *mtx*

$([a1, b1] \#$   
 $[c1, d1] \# []) +$  *mtx*

$([a2, b2] \#$   
 $[c2, d2] \# []) =$  ((*mtx*

$[a1+a2, b1+b2] \#$   
 $[c1+c2, d1+d2] \# [])::2$  *sq-mtx*)

**by** (*simp add: sq-mtx-eq-iff*)

**lemma** *minus-mtx2*: *mtx*

$([a1, b1] \#$   
 $[c1, d1] \# []) -$  *mtx*

$([a2, b2] \#$   
 $[c2, d2] \# []) =$  ((*mtx*

$[a1-a2, b1-b2] \#$   
 $[c1-c2, d1-d2] \# [])::2$  *sq-mtx*)

**by** (*simp add: sq-mtx-eq-iff*)

**lemma** *times-mtx2*: *mtx*

$([a1, b1] \#$   
 $[c1, d1] \# []) *$  *mtx*

$([a2, b2] \#$   
 $[c2, d2] \# []) =$  ((*mtx*

$[a1*a2+b1*c2, a1*b2+b1*d2] \#$   
 $[c1*a2+d1*c2, c1*b2+d1*d2] \# [])::2$  *sq-mtx*)

**unfolding** *sq-mtx-times-eq UNIV-2*

**by** (*simp add: sq-mtx-eq-iff*)

**3x3 matrices****lemma** *mtx3-to-mtx*: *mtx*

```

([a11, a12, a13] #
 [a21, a22, a23] #
 [a31, a32, a33] # []) =
to-mtx (λ i j::3. if i=1 ∧ j=1 then a11
 else if i=1 ∧ j=2 then a12
 else if i=1 ∧ j=3 then a13
 else if i=2 ∧ j=1 then a21
 else if i=2 ∧ j=2 then a22
 else if i=2 ∧ j=3 then a23
 else if i=3 ∧ j=1 then a31
 else if i=3 ∧ j=2 then a32
 else a33))))))
apply(simp add: sq-mtx-eq-iff)
using exhaust-3 by force

```

**abbreviation** *diag3* :: *real* ⇒ *real* ⇒ *real* ⇒ 3 *sq-mtx***where** *diag3* *ι*<sub>1</sub> *ι*<sub>2</sub> *ι*<sub>3</sub> ≡ *mtx*

```

([ι1, 0, 0] #
 [0, ι2, 0] #
 [0, 0, ι3] # [])

```

**lemma** *diag3-eq*: *diag3* (*ι* 1) (*ι* 2) (*ι* 3) = (diag *i*. *ι* *i*)

```

apply(simp add: sq-mtx-eq-iff)
using exhaust-3 by (force simp: axis-def)

```

**lemma** *one-mtx3*: (1::3 *sq-mtx*) = *diag3* 1 1 1

```

apply(subst sq-mtx-eq-iff)
using exhaust-3 by force

```

**lemma** *zero-mtx3*: (0::3 *sq-mtx*) = *diag3* 0 0 0**by** (simp add: sq-mtx-eq-iff)**lemma** *scaleR-mtx3*: *k* \*<sub>R</sub> *mtx*

```

([k*a11, k*a12, k*a13] #
 [k*a21, k*a22, k*a23] #
 [k*a31, k*a32, k*a33] # []) = mtx
by (simp add: sq-mtx-eq-iff)

```

**lemma** *plus-mtx3*: *mtx*

```

([a11, a12, a13] #
 [a21, a22, a23] #
 [a31, a32, a33] # []) + mtx
([b11, b12, b13] #
 [b21, b22, b23] #
 [b31, b32, b33] # []) = (mtx
 [a11+b11, a12+b12, a13+b13] #
 [a21+b21, a22+b22, a23+b23] #
 [a31+b31, a32+b32, a33+b33] # [])::3 sq-mtx)
by (subst sq-mtx-eq-iff) simp

```

**lemma** *minus-mtx3*: *mtx*

```

([a11, a12, a13] #
 [a21, a22, a23] #
 [a31, a32, a33] # []) - mtx

```

```

([b11, b12, b13] #
 [b21, b22, b23] #
 [b31, b32, b33] # []) = (mtx
 ([a11-b11, a12-b12, a13-b13] #
 [a21-b21, a22-b22, a23-b23] #
 [a31-b31, a32-b32, a33-b33] # [])::3 sq-mtx)
by (simp add: sq-mtx-eq-iff)

```

**lemma** *times-mtx3*: *mtx*

```

([a11, a12, a13] #
 [a21, a22, a23] #
 [a31, a32, a33] # []) * mtx
([b11, b12, b13] #
 [b21, b22, b23] #
 [b31, b32, b33] # []) = (mtx
 ([a11*b11+a12*b21+a13*b31, a11*b12+a12*b22+a13*b32, a11*b13+a12*b23+a13*b33] #
 [a21*b11+a22*b21+a23*b31, a21*b12+a22*b22+a23*b32, a21*b13+a22*b23+a23*b33] #
 [a31*b11+a32*b21+a33*b31, a31*b12+a32*b22+a33*b32, a31*b13+a32*b23+a33*b33] # [])::3 sq-mtx)
unfolding sq-mtx-times-eq
unfolding UNIV-3 by (simp add: sq-mtx-eq-iff)

```

**end**

## 0.14 Affine systems of ODEs

Affine systems of ordinary differential equations (ODEs) are those whose vector fields are linear operators. Broadly speaking, if there are functions  $A$  and  $B$  such that the system of ODEs  $X't = f(Xt)$  turns into  $X't = (At) \cdot (Xt) + (Bt)$ , then it is affine. The end goal of this section is to prove that every affine system of ODEs has a unique solution, and to obtain a characterization of said solution.

**theory** *MTX-Flows*

**imports** *SQ-MTX ../HS-ODEs*

**begin**

### 0.14.1 Existence and uniqueness for affine systems

**definition** *matrix-continuous-on* :: *real set*  $\Rightarrow$  (*real*  $\Rightarrow$  (*'a::real-normed-algebra-1*)  $\wedge$  *'n*  $\wedge$  *'m*)  $\Rightarrow$  *bool*  
**where** *matrix-continuous-on*  $T A = (\forall t \in T. \forall \varepsilon > 0. \exists \delta > 0. \forall \tau \in T. |\tau - t| < \delta \longrightarrow \|A \tau - A t\|_{op} \leq \varepsilon)$

**lemma** *continuous-on-matrix-vector-multl*:

**assumes** *matrix-continuous-on*  $T A$

**shows** *continuous-on*  $T (\lambda t. A t * v s)$

**proof**(*rule continuous-onI, simp add: dist-norm*)

**fix**  $e t::real$  **assume**  $0 < e$  **and**  $t \in T$

**let**  $? \varepsilon = e / (\|(if s = 0 then 1 else s)\|)$

**have**  $? \varepsilon > 0$

**using**  $\langle 0 < e \rangle$  **by** *simp*

**then obtain**  $\delta$  **where** *dHyp*:  $\delta > 0 \wedge (\forall \tau \in T. |\tau - t| < \delta \longrightarrow \|A \tau - A t\|_{op} \leq ? \varepsilon)$

**using** *assms*  $\langle t \in T \rangle$  **unfolding** *dist-norm matrix-continuous-on-def* **by** *fastforce*

**{fix**  $\tau$  **assume**  $\tau \in T$  **and**  $|\tau - t| < \delta$

**have** *obs*:  $? \varepsilon * (\|s\|) = (if s = 0 then 0 else e)$

**by** *auto*

**have**  $\|A \tau * v s - A t * v s\| = \|(A \tau - A t) * v s\|$

**by** (*simp add: matrix-vector-mult-diff-rdistrib*)

**also have**  $\dots \leq (\|A \tau - A t\|_{op}) * (\|s\|)$

**using** *norm-matrix-le-mult-op-norm* **by** *blast*

also have ...  $\leq ?\varepsilon * (\|s\|)$   
 using *dHyp*  $\langle \tau \in T \rangle \langle |\tau - t| < \delta \rangle$  *mult-right-mono norm-ge-zero* **by** *blast*  
 finally have  $\|A \tau * v s - A t * v s\| \leq e$   
 by (*subst (asm) obs*) (*metis (mono-tags, hide-lams)*  $\langle 0 < e \rangle$  *less-eq-real-def order-trans*)  
 thus  $\exists d > 0. \forall \tau \in T. |\tau - t| < d \longrightarrow \|A \tau * v s - A t * v s\| \leq e$   
 using *dHyp* **by** *blast*  
**qed**

**lemma** *lipschitz-cond-affine*:

fixes  $A :: \text{real} \Rightarrow 'a::\text{real-normed-algebra-1}^n^m$  **and**  $T::\text{real set}$   
 defines  $L \equiv \text{Sup } \{\|A t\|_{op} \mid t. t \in T\}$   
 assumes  $t \in T$  **and** *bdd-above*  $\{\|A t\|_{op} \mid t. t \in T\}$   
 shows  $\|A t * v x - A t * v y\| \leq L * (\|x - y\|)$

**proof**–

have *obs*:  $\|A t\|_{op} \leq \text{Sup } \{\|A t\|_{op} \mid t. t \in T\}$   
 apply(*rule cSup-upper*)  
 using *continuous-on-subset assms* **by** (*auto simp: dist-norm*)  
 have  $\|A t * v x - A t * v y\| = \|A t * v (x - y)\|$   
 by (*simp add: matrix-vector-mult-diff-distrib*)  
 also have ...  $\leq (\|A t\|_{op}) * (\|x - y\|)$   
 using *norm-matrix-le-mult-op-norm* **by** *blast*  
 also have ...  $\leq \text{Sup } \{\|A t\|_{op} \mid t. t \in T\} * (\|x - y\|)$   
 using *obs mult-right-mono norm-ge-zero* **by** *blast*  
 finally show  $\|A t * v x - A t * v y\| \leq L * (\|x - y\|)$   
 unfolding *assms* .

**qed**

**lemma** *local-lipschitz-affine*:

fixes  $A :: \text{real} \Rightarrow 'a::\text{real-normed-algebra-1}^n^m$   
 assumes *open*  $T$  **and** *open*  $S$   
 and *Ahyp*:  $\bigwedge \tau \varepsilon. \varepsilon > 0 \implies \tau \in T \implies \text{cball } \tau \varepsilon \subseteq T \implies \text{bdd-above } \{\|A t\|_{op} \mid t. t \in \text{cball } \tau \varepsilon\}$   
 shows *local-lipschitz*  $T S (\lambda t s. A t * v s + B t)$

**proof**(*unfold local-lipschitz-def lipschitz-on-def, clarsimp*)

fix  $s t$  assume  $s \in S$  **and**  $t \in T$   
 then obtain  $e1 e2$  where  $\text{cball } t e1 \subseteq T$  **and**  $\text{cball } s e2 \subseteq S$  **and**  $\min e1 e2 > 0$   
 using *open-cballE*[*OF* - *open T*] *open-cballE*[*OF* - *open S*] **by** *force*  
 hence *obs*:  $\text{cball } t (\min e1 e2) \subseteq T$   
 by *auto*  
 let  $?L = \text{Sup } \{\|A \tau\|_{op} \mid \tau. \tau \in \text{cball } t (\min e1 e2)\}$   
 have  $\|A t\|_{op} \in \{\|A \tau\|_{op} \mid \tau. \tau \in \text{cball } t (\min e1 e2)\}$   
 using  $\langle \min e1 e2 > 0 \rangle$  **by** *auto*  
 moreover have *bdd*: *bdd-above*  $\{\|A \tau\|_{op} \mid \tau. \tau \in \text{cball } t (\min e1 e2)\}$   
 by (*rule Ahyp, simp only: min e1 e2 > 0, simp-all add: t ∈ T obs*)  
 moreover have  $\text{Sup } \{\|A \tau\|_{op} \mid \tau. \tau \in \text{cball } t (\min e1 e2)\} \geq 0$   
 apply(*rule order.trans*[*OF op-norm-ge-0* [*of A t*]])  
 by (*rule cSup-upper* [*OF calculation*])  
 moreover have  $\forall x \in \text{cball } s (\min e1 e2) \cap S. \forall y \in \text{cball } s (\min e1 e2) \cap S.$   
 $\forall \tau \in \text{cball } t (\min e1 e2) \cap T. \text{dist } (A \tau * v x) (A \tau * v y) \leq ?L * \text{dist } x y$   
 apply(*clarify, simp only: dist-norm, rule lipschitz-cond-affine*)  
 using  $\langle \min e1 e2 > 0 \rangle$  *bdd* **by** *auto*  
 ultimately show  $\exists e > 0. \exists L. \forall t \in \text{cball } t e \cap T. 0 \leq L \wedge$   
 $(\forall x \in \text{cball } s e \cap S. \forall y \in \text{cball } s e \cap S. \text{dist } (A t * v x) (A t * v y) \leq L * \text{dist } x y)$   
 using  $\langle \min e1 e2 > 0 \rangle$  **by** *blast*

**qed**

**lemma** *picard-lindelof-affine*:

fixes  $A :: \text{real} \Rightarrow 'a::\{\text{banach, real-normed-algebra-1, heine-borel}\}^n^n$   
 assumes *Ahyp*: *matrix-continuous-on*  $T A$   
 and  $\bigwedge \tau \varepsilon. \tau \in T \implies \varepsilon > 0 \implies \text{bdd-above } \{\|A t\|_{op} \mid t. \text{dist } \tau t \leq \varepsilon\}$   
 and *Bhyp*: *continuous-on*  $T B$  **and** *open*  $S$

**and**  $t_0 \in T$  **and** *Thyp*: open  $T$  is-interval  $T$   
**shows** *picard-lindelof*  $(\lambda t s. A * v s + B t) T S t_0$   
**apply** (*unfold-locales*, *simp-all add: assms*, *clarsimp*)  
**apply** (*rule continuous-on-add*[*OF continuous-on-matrix-vector-multl*[*OF Ahyp*] *Bhyp*])  
**by** (*rule local-lipschitz-affine*) (*simp-all add: assms*)

**lemma** *picard-lindelof-autonomous-affine*:  
**fixes**  $A :: 'a::\{\text{banach, real-normed-field, heine-borel}\}^n^n$   
**shows** *picard-lindelof*  $(\lambda t s. A * v s + B) UNIV UNIV t_0$   
**using** *picard-lindelof-affine*[*of -  $\lambda t. A \lambda t. B$* ]  
**unfolding** *matrix-continuous-on-def* **by** (*simp only: diff-self op-norm0, auto*)

**lemma** *picard-lindelof-autonomous-linear*:  
**fixes**  $A :: 'a::\{\text{banach, real-normed-field, heine-borel}\}^n^n$   
**shows** *picard-lindelof*  $(\lambda t. (*v) A) UNIV UNIV t_0$   
**using** *picard-lindelof-autonomous-affine*[*of A 0*] **by** *force*

**lemmas** *unique-sol-autonomous-affine* = *picard-lindelof.ivp-unique-solution*[*OF picard-lindelof-autonomous-affine UNIV-I - subset-UNIV*]

**lemmas** *unique-sol-autonomous-linear* = *picard-lindelof.ivp-unique-solution*[*OF picard-lindelof-autonomous-linear UNIV-I - subset-UNIV*]

### 0.14.2 Flow for affine systems

#### Derivative rules for square matrices

**lemma** *has-derivative-exp-scaleRl*[*derivative-intros*]:  
**fixes**  $f::\text{real} \Rightarrow \text{real}$   
**assumes**  $D f \mapsto f'$  at  $t$  within  $T$   
**shows**  $D (\lambda t. \exp (f t *_R A)) \mapsto (\lambda h. f' h *_R (\exp (f t *_R A) * A))$  at  $t$  within  $T$   
**proof** –  
**have** *bounded-linear*  $f'$   
**using** *assms* **by** *auto*  
**then obtain**  $m$  **where**  $\text{obs}: f' = (\lambda h. h * m)$   
**using** *real-bounded-linear* **by** *blast*  
**thus** *?thesis*  
**using** *vector-diff-chain-within*[*OF - exp-scaleR-has-vector-derivative-right*]  
*assms obs* **by** (*auto simp: has-vector-derivative-def comp-def*)  
**qed**

**lemma** *vderiv-on-exp-scaleRI*[*poly-derivatives*]:  
**assumes**  $D f = f'$  on  $T$  **and**  $g' = (\lambda x. f' x *_R \exp (f x *_R A) * A)$   
**shows**  $D (\lambda x. \exp (f x *_R A)) = g'$  on  $T$   
**using** *assms* **unfolding** *has-vderiv-on-def* *has-vector-derivative-def* **apply** *clarsimp*  
**by** (*rule has-derivative-exp-scaleRl, auto simp: fun-eq-iff*)

**lemma** *has-derivative-mtx-ith*[*derivative-intros*]:  
**fixes**  $t::\text{real}$  **and**  $T :: \text{real set}$   
**defines**  $t_0 \equiv \text{netlimit (at } t \text{ within } T)$   
**assumes**  $D A \mapsto (\lambda h. h *_R A' t)$  at  $t$  within  $T$   
**shows**  $D (\lambda t. A t \$\$ i) \mapsto (\lambda h. h *_R A' t \$\$ i)$  at  $t$  within  $T$   
**using** *assms* **unfolding** *has-derivative-def* **apply** *safe*  
**apply**(*force simp: bounded-linear-def bounded-linear-axioms-def*)  
**apply**(*rule-tac F= $\lambda \tau. (A \tau - A t_0 - (\tau - t_0) *_R A' t) /_R (\|\tau - t_0\|)$  in tendsto-zero-norm-bound*)  
**by** (*clarsimp, rule mult-left-mono, metis (no-types, lifting) norm-column-le-norm sq-mtx-minus-ith sq-mtx-scaleR-ith simp-all*)

**lemmas** *has-derivative-mtx-vec-mult*[*derivative-intros*] =  
*bounded-bilinear.FDERIV*[*OF bounded-bilinear-sq-mtx-vec-mult*]

**lemma** *vderiv-on-mtx-vec-multl*[poly-derivatives]:  
**assumes**  $D\ u = u'$  on  $T$  **and**  $D\ A = A'$  on  $T$   
**and**  $g = (\lambda t. A\ t *_{\mathcal{V}} u' t + A' t *_{\mathcal{V}} u t)$   
**shows**  $D\ (\lambda t. A\ t *_{\mathcal{V}} u t) = g$  on  $T$   
**using** *assms unfolding has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*  
**apply**(*erule-tac x=x in ballE, simp-all*) +  
**apply**(*rule derivative-eq-intros(142)*)  
**by** (*auto simp: fun-eq-iff mtx-vec-scaleR-commute pth-6 scaleR-mtx-vec-assoc*)

**lemmas** *has-vderiv-on-ivl-integral = ivl-integral-has-vderiv-on*[*OF vderiv-on-continuous-on*]

**declare** *has-vderiv-on-ivl-integral* [poly-derivatives]

**lemma** *has-derivative-mtx-vec-multl*[derivative-intros]:  
**assumes**  $\bigwedge i\ j. D\ (\lambda t. (A\ t) \$\$ i\ \$ j) \mapsto (\lambda \tau. \tau *_{\mathcal{R}} (A' t) \$\$ i\ \$ j)$  (*at t within T*)  
**shows**  $D\ (\lambda t. A\ t *_{\mathcal{V}} x) \mapsto (\lambda \tau. \tau *_{\mathcal{R}} (A' t) *_{\mathcal{V}} x)$  *at t within T*  
**unfolding** *sq-mtx-vec-mult-sum-cols*  
**apply**(*rule-tac f'1= $\lambda i\ \tau. \tau *_{\mathcal{R}} (x \$ i *_{\mathcal{R}} \text{col } i (A' t))$  in derivative-eq-intros(10)*)  
**apply**(*simp-all add: scaleR-right.sum*)  
**apply**(*rule-tac g'1= $\lambda \tau. \tau *_{\mathcal{R}} \text{col } i (A' t)$  in derivative-eq-intros(4), simp-all add: mult.commute*)  
**using** *assms unfolding sq-mtx-col-def column-def* **apply**(*transfer, simp*)  
**apply**(*rule has-derivative-vec-lambda*)  
**by** (*simp add: scaleR-vec-def*)

**lemma** *continuous-on-mtx-vec-multl*: *continuous-on S (( $*_{\mathcal{V}}$ ) A)*  
**by** *transfer (simp add: matrix-vector-mult-linear-continuous-on)*

— Automatically generated derivative rules from this subsubsection

**thm** *derivative-eq-intros(140,141,142,143)*

## Existence and uniqueness with square matrices

Finally, we can use the *exp* operation to characterize the general solutions for affine systems of ODEs. We show that they satisfy the *local-flow* locale.

**lemma** *continuous-on-sq-mtx-vec-multl*:  
**fixes**  $A :: \text{real} \Rightarrow ('n::\text{finite})\ \text{sq-mtx}$   
**assumes** *continuous-on T A*  
**shows** *continuous-on T ( $\lambda t. A\ t *_{\mathcal{V}} s$ )*  
**proof—**  
**have** *matrix-continuous-on T ( $\lambda t. \text{to-vec } (A\ t)$ )*  
**using** *assms by (force simp: continuous-on-iff dist-norm norm-sq-mtx-def matrix-continuous-on-def)*  
**hence** *continuous-on T ( $\lambda t. \text{to-vec } (A\ t) *_{\mathcal{V}} s$ )*  
**by** (*rule continuous-on-matrix-vector-multl*)  
**thus** *?thesis*  
**by** *transfer*  
**qed**

**lemmas** *continuous-on-affine = continuous-on-add*[*OF continuous-on-sq-mtx-vec-multl*]

**lemma** *local-lipschitz-sq-mtx-affine*:  
**fixes**  $A :: \text{real} \Rightarrow ('n::\text{finite})\ \text{sq-mtx}$   
**assumes** *continuous-on T A open T open S*  
**shows** *local-lipschitz T S ( $\lambda t\ s. A\ t *_{\mathcal{V}} s + B\ t$ )*  
**proof—**  
**have** *obs:  $\bigwedge \tau\ \varepsilon. 0 < \varepsilon \implies \tau \in T \implies \text{cball } \tau\ \varepsilon \subseteq T \implies \text{bdd-above } \{\|A\ t\| \mid t \in \text{cball } \tau\ \varepsilon\}$*   
**by** (*rule bdd-above-norm-cont-comp, rule continuous-on-subset[OF assms(1)], simp-all*)  
**hence**  $\bigwedge \tau\ \varepsilon. 0 < \varepsilon \implies \tau \in T \implies \text{cball } \tau\ \varepsilon \subseteq T \implies \text{bdd-above } \{\|\text{to-vec } (A\ t)\|_{op} \mid t \in \text{cball } \tau\ \varepsilon\}$

by (simp add: norm-sq-mtx-def)  
 hence local-lipschitz  $T\ S\ (\lambda t\ s.\ \text{to-vec}\ (A\ t)\ *v\ s + B\ t)$   
 using local-lipschitz-affine[OF assms(2,3), of  $\lambda t.\ \text{to-vec}\ (A\ t)$ ] by force  
 thus ?thesis  
 by transfer  
 qed

lemma picard-lindelof-sq-mtx-affine:  
 assumes continuous-on  $T\ A$  and continuous-on  $T\ B$   
 and  $t_0 \in T$  is-interval  $T$  open  $T$  and open  $S$   
 shows picard-lindelof  $(\lambda t\ s.\ A\ t *v\ s + B\ t)\ T\ S\ t_0$   
 apply(unfold-locales, simp-all add: assms, clarsimp)  
 using continuous-on-affine assms apply blast  
 by (rule local-lipschitz-sq-mtx-affine, simp-all add: assms)

lemmas sq-mtx-unique-sol-autonomous-affine = picard-lindelof.ivep-unique-solution[OF  
 picard-lindelof-sq-mtx-affine[OF  
 continuous-on-const  
 continuous-on-const  
 UNIV-I is-interval-univ  
 open-UNIV open-UNIV]  
 UNIV-I - subset-UNIV]

lemma has-vderiv-on-sq-mtx-linear:  
 $D\ (\lambda t.\ \exp((t - t_0) *R\ A) *v\ s) = (\lambda t.\ A *v\ (\exp((t - t_0) *R\ A) *v\ s))$  on  $\{t_0 \dots t\}$   
 by (rule poly-derivatives)+ (auto simp: exp-times-scaleR-commute sq-mtx-times-vec-assoc)

lemma has-vderiv-on-sq-mtx-affine:  
 fixes  $t_0::\text{real}$  and  $A :: ('a::\text{finite})\ \text{sq-mtx}$   
 defines  $\text{lSol}\ c\ t \equiv \exp((c * (t - t_0)) *R\ A)$   
 shows  $D\ (\lambda t.\ \text{lSol}\ 1\ t *v\ s + \text{lSol}\ 1\ t *v\ (\int_{t_0}^t (\text{lSol}\ (-1)\ \tau *v\ B)\ \partial\tau)) =$   
 $(\lambda t.\ A *v\ (\text{lSol}\ 1\ t *v\ s + \text{lSol}\ 1\ t *v\ (\int_{t_0}^t (\text{lSol}\ (-1)\ \tau *v\ B)\ \partial\tau)) + B)$  on  $\{t_0 \dots t\}$   
 unfolding assms apply(simp only: mult.left-neutral mult-minus1)  
 apply(rule poly-derivatives, (force)?, (force)?, (force)?, (force)?)+  
 by (simp add: mtx-vec-mult-add-rdistl sq-mtx-times-vec-assoc[symmetric]  
 exp-minus-inverse exp-times-scaleR-commute mult-exp-exp scale-left-distrib[symmetric])

lemma autonomous-linear-sol-is-exp:  
 assumes  $D\ X = (\lambda t.\ A *v\ X\ t)$  on  $\{t_0 \dots t\}$  and  $X\ t_0 = s$   
 shows  $X\ t = \exp((t - t_0) *R\ A) *v\ s$   
 apply(rule sq-mtx-unique-sol-autonomous-affine[of  $\lambda s.\ \{t_0 \dots t\} - t\ X\ A\ 0$ ])  
 using assms apply(simp-all add: ivep-sols-def)  
 using has-vderiv-on-sq-mtx-linear by force+

lemma autonomous-affine-sol-is-exp-plus-int:  
 assumes  $D\ X = (\lambda t.\ A *v\ X\ t + B)$  on  $\{t_0 \dots t\}$  and  $X\ t_0 = s$   
 shows  $X\ t = \exp((t - t_0) *R\ A) *v\ s + \exp((t - t_0) *R\ A) *v\ (\int_{t_0}^t (\exp(-(t - \tau) *R\ A) *v\ B)\ \partial\tau)$   
 apply(rule sq-mtx-unique-sol-autonomous-affine[of  $\lambda s.\ \{t_0 \dots t\} - t\ X\ A\ B$ ])  
 using assms apply(simp-all add: ivep-sols-def)  
 using has-vderiv-on-sq-mtx-affine by force+

lemma local-flow-sq-mtx-linear: local-flow  $((*v)\ A)\ UNIV\ UNIV\ (\lambda t\ s.\ \exp(t *R\ A) *v\ s)$   
 unfolding local-flow-def local-flow-axioms-def apply safe  
 using picard-lindelof-sq-mtx-affine[of  $\lambda t.\ A\ \lambda t.\ 0$ ] apply force  
 using has-vderiv-on-sq-mtx-linear[of 0] by auto

lemma local-flow-sq-mtx-affine: local-flow  $(\lambda s.\ A *v\ s + B)\ UNIV\ UNIV\$   
 $(\lambda t\ s.\ \exp(t *R\ A) *v\ s + \exp(t *R\ A) *v\ (\int_0^t (\exp(-\tau *R\ A) *v\ B)\ \partial\tau))$   
 unfolding local-flow-def local-flow-axioms-def apply safe  
 using picard-lindelof-sq-mtx-affine[of  $\lambda t.\ A\ \lambda t.\ B$ ] apply force



```

using has-vderiv-on-sq-mtx-affine[of 0 A] by auto

end

```

## 0.15 Verification examples

```

theory MTX-Examples
imports MTX-Flows ../HS-VC-Spartan

```

```

begin

```

### 0.15.1 Examples

```

abbreviation hoareT :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  ('a  $\Rightarrow$  'a set)  $\Rightarrow$  ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
  (PRE- HP - POST - [85,85]85) where PRE P HP X POST Q  $\equiv$  (P  $\leq$  [X] Q)

```

**Verification by uniqueness.**

```

abbreviation mtx-circ :: 2 sq-mtx (A)
where A  $\equiv$  mtx
  ([0, 1] #
   [-1, 0] # [])

```

```

abbreviation mtx-circ-flow :: real  $\Rightarrow$  real2  $\Rightarrow$  real2 ( $\varphi$ )
where  $\varphi$  t s  $\equiv$  ( $\chi$  i. if i = 1 then s$1 * cos t + s$2 * sin t else - s$1 * sin t + s$2 * cos t)

```

```

lemma mtx-circ-flow-eq: exp (t *R A) *V s =  $\varphi$  t s
apply(rule local-flow.eq-solution[OF local-flow-sq-mtx-linear, symmetric, of -  $\lambda$ s. UNIV], simp-all)
apply(rule ivp-solsI, simp-all add: sq-mtx-vec-mult-eq vec-eq-iff)
unfolding UNIV-2 using exhaust-2
by (force intro!: poly-derivatives simp: matrix-vector-mult-def)+

```

```

lemma mtx-circ:
  PRE( $\lambda$ s. r2 = (s $ 1)2 + (s $ 2)2)
  HP x' = (*V) A & G
  POST ( $\lambda$ s. r2 = (s $ 1)2 + (s $ 2)2)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-sq-mtx-linear])
unfolding mtx-circ-flow-eq by auto

```

```

no-notation mtx-circ (A)
and mtx-circ-flow ( $\varphi$ )

```

**Flow of diagonalisable matrix.**

```

abbreviation mtx-hOsc :: real  $\Rightarrow$  real  $\Rightarrow$  2 sq-mtx (A)
where A a b  $\equiv$  mtx
  ([0, 1] #
   [a, b] # [])

```

```

abbreviation mtx-chB-hOsc :: real  $\Rightarrow$  real  $\Rightarrow$  2 sq-mtx (P)
where P a b  $\equiv$  mtx
  ([a, b] #
   [1, 1] # [])

```

```

lemma inv-mtx-chB-hOsc:
  a  $\neq$  b  $\implies$  (P a b)-1 = (1/(a - b)) *R mtx
  ([ 1, -b] #
   [-1, a] # [])
apply(rule sq-mtx-inv-unique, unfold scaleR-mtx2 times-mtx2)

```

by (simp add: diff-divide-distrib[symmetric] one-mtx2)+

**lemma** invertible-mtx-chB-hOsc:  $a \neq b \implies \text{mtx-invertible } (P \ a \ b)$   
**apply**(rule mtx-invertibleI[of - (P a b)<sup>-1</sup>])  
**apply**(unfold inv-mtx-chB-hOsc scaleR-mtx2 times-mtx2 one-mtx2)  
 by (subst sq-mtx-eq-iff, simp add: vector-def frac-diff-eq1)+

**lemma** mtx-hOsc-diagonalizable:

**fixes**  $a \ b :: \text{real}$   
**defines**  $\iota_1 \equiv (b - \text{sqrt } (b^2 + 4 * a)) / 2$  **and**  $\iota_2 \equiv (b + \text{sqrt } (b^2 + 4 * a)) / 2$   
**assumes**  $b^2 + a * 4 > 0$  **and**  $a \neq 0$   
**shows**  $A \ a \ b = P \ (-\iota_2 / a) \ (-\iota_1 / a) * (\text{diag } i. \text{ if } i = 1 \text{ then } \iota_1 \text{ else } \iota_2) * (P \ (-\iota_2 / a) \ (-\iota_1 / a))^{-1}$   
**unfolding** *assms* **apply**(subst inv-mtx-chB-hOsc)  
**using** *assms*(3,4) **apply**(simp-all add: diag2-eq[symmetric])  
**unfolding** sq-mtx-times-eq sq-mtx-scaleR-eq UNIV-2 **apply**(subst sq-mtx-eq-iff)  
**using** exhaust-2 *assms* **by** (auto simp: field-simps, auto simp: field-power-simps)

**lemma** mtx-hOsc-solution-eq:

**fixes**  $a \ b :: \text{real}$   
**defines**  $\iota_1 \equiv (b - \text{sqrt } (b^2 + 4 * a)) / 2$  **and**  $\iota_2 \equiv (b + \text{sqrt } (b^2 + 4 * a)) / 2$   
**defines**  $\Phi \ t \equiv \text{mtx } ([\iota_2 * \exp(t * \iota_1) - \iota_1 * \exp(t * \iota_2), \exp(t * \iota_2) - \exp(t * \iota_1)] \# [a * \exp(t * \iota_2) - a * \exp(t * \iota_1), \iota_2 * \exp(t * \iota_2) - \iota_1 * \exp(t * \iota_1)] \# [])$   
**assumes**  $b^2 + a * 4 > 0$  **and**  $a \neq 0$   
**shows**  $P \ (-\iota_2 / a) \ (-\iota_1 / a) * (\text{diag } i. \exp(t * (\text{if } i = 1 \text{ then } \iota_1 \text{ else } \iota_2))) * (P \ (-\iota_2 / a) \ (-\iota_1 / a))^{-1} = (1 / \text{sqrt } (b^2 + a * 4)) *_{\text{R}} (\Phi \ t)$   
**unfolding** *assms* **apply**(subst inv-mtx-chB-hOsc)  
**using** *assms* **apply**(simp-all add: mtx-times-scaleR-commute, subst sq-mtx-eq-iff)  
**unfolding** UNIV-2 sq-mtx-times-eq sq-mtx-scaleR-eq sq-mtx-uminus-eq **apply**(simp-all add: axis-def)  
**by** (auto simp: field-simps, auto simp: field-power-simps)+

**lemma** local-flow-mtx-hOsc:

**fixes**  $a \ b$   
**defines**  $\iota_1 \equiv (b - \text{sqrt } (b^2 + 4 * a)) / 2$  **and**  $\iota_2 \equiv (b + \text{sqrt } (b^2 + 4 * a)) / 2$   
**defines**  $\Phi \ t \equiv \text{mtx } ([\iota_2 * \exp(t * \iota_1) - \iota_1 * \exp(t * \iota_2), \exp(t * \iota_2) - \exp(t * \iota_1)] \# [a * \exp(t * \iota_2) - a * \exp(t * \iota_1), \iota_2 * \exp(t * \iota_2) - \iota_1 * \exp(t * \iota_1)] \# [])$   
**assumes**  $b^2 + a * 4 > 0$  **and**  $a \neq 0$   
**shows**  $\text{local-flow } ((*_V) \ (A \ a \ b)) \ \text{UNIV} \ \text{UNIV} \ (\lambda t. (*_V) \ ((1 / \text{sqrt } (b^2 + a * 4)) *_{\text{R}} \Phi \ t))$   
**unfolding** *assms* **using** local-flow-sq-mtx-linear[of A a b] *assms*  
**apply**(subst (asm) exp-scaleR-diagonal2[OF invertible-mtx-chB-hOsc mtx-hOsc-diagonalizable])  
**apply**(simp, simp, simp)  
**by** (subst (asm) mtx-hOsc-solution-eq) simp-all

**lemma** overdamped-door-arith:

**assumes**  $b^2 + a * 4 > 0$  **and**  $a < 0$  **and**  $b \leq 0$  **and**  $t \geq 0$  **and**  $s1 > 0$   
**shows**  $0 \leq ((b + \text{sqrt } (b^2 + 4 * a)) * \exp(t * (b - \text{sqrt } (b^2 + 4 * a)) / 2) / 2 - (b - \text{sqrt } (b^2 + 4 * a)) * \exp(t * (b + \text{sqrt } (b^2 + 4 * a)) / 2) / 2) * s1 / \text{sqrt } (b^2 + a * 4)$   
**proof**(subst diff-divide-distrib[symmetric], simp)  
**have**  $f0: s1 / (2 * \text{sqrt } (b^2 + a * 4)) > 0$  **(is**  $s1 / ?c3 > 0$ **)**  
**using** *assms*(1,5) **by** simp  
**have**  $f1: (b - \text{sqrt } (b^2 + 4 * a)) < (b + \text{sqrt } (b^2 + 4 * a))$  **(is**  $?c2 < ?c1$ **)**  
**and**  $f2: (b + \text{sqrt } (b^2 + 4 * a)) < 0$   
**using** sqrt-ge-absD[of b b<sup>2</sup> + 4 \* a] *assms* **by** (force, linarith)  
**hence**  $f3: \exp(t * ?c2 / 2) \leq \exp(t * ?c1 / 2)$  **(is**  $\exp ?t1 \leq \exp ?t2$ **)**  
**unfolding** exp-le-cancel-iff  
**using** *assms*(4) **by** (case-tac t=0, simp-all)  
**hence**  $?c2 * \exp ?t2 \leq ?c2 * \exp ?t1$   
**using** f1 f2 real-mult-le-cancel-iff2[of -?c2 exp ?t1 exp ?t2] **by** linarith  
**also have**  $\dots < ?c1 * \exp ?t1$

```

using f1 by auto
also have... ≤ ?c1 * exp ?t1
using f1 f2 by auto
ultimately show 0 ≤ (?c1 * exp ?t1 - ?c2 * exp ?t2) * s1 / ?c3
using f0 f1 asms(5) by auto
qed

```

abbreviation  $\text{open-door } s \equiv \{s. s\$1 > 0 \wedge s\$2 = 0\}$

lemma *overdamped-door*:

```

assumes  $b^2 + a * 4 > 0$  and  $a < 0$  and  $b \leq 0$ 
shows  $PRE (\lambda s. s\$1 = 0)$ 
HP (LOOP open-door; ( $x' = (*_V) (A \ a \ b)$ ) & G) INV ( $\lambda s. 0 \leq s\$1$ )
POST ( $\lambda s. 0 \leq s \$ 1$ )
apply(rule fbox-loopI, simp-all add: le-fun-def)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-mtx-hOsc[OF asms(1)]])
using asms apply(simp-all add: le-fun-def fbox-def)
unfolding sq-mtx-scaleR-eq UNIV-2 sq-mtx-vec-mult-eq
by (clarsimp simp: overdamped-door-arith)

```

no-notation *mtx-hOsc* ( $A$ )  
and *mtx-chB-hOsc* ( $P$ )

### Flow of non-diagonalisable matrix.

abbreviation *mtx-cnst-acc* ::  $3 \text{ sq-mtx } (K)$

```

where  $K \equiv \text{mtx } ($ 
 $[0, 1, 0] \#$ 
 $[0, 0, 1] \#$ 
 $[0, 0, 0] \# [])$ 

```

lemma *pow2-scaleR-mtx-cnst-acc*:  $(t *_R K)^2 = \text{mtx } ($   
 $[0, 0, t^2] \#$   
 $[0, 0, 0] \#$   
 $[0, 0, 0] \# [])$

```

unfolding power2-eq-square apply(subst sq-mtx-eq-iff)
unfolding sq-mtx-times-eq UNIV-3 by auto

```

lemma *powN-scaleR-mtx-cnst-acc*:  $n > 2 \implies (t *_R K)^n = 0$

```

apply(induct n, simp, case-tac  $n \leq 2$ )
apply(subgoal-tac  $n = 2$ , erule ssubst)
unfolding power-Suc2 pow2-scaleR-mtx-cnst-acc sq-mtx-times-eq UNIV-3
by (auto simp: sq-mtx-eq-iff)

```

lemma *exp-mtx-cnst-acc*:  $\exp(t *_R K) = ((t *_R K)^2 /_R 2) + (t *_R K) + 1$

```

unfolding exp-def apply(subst suminf-eq-sum[of 2])
using powN-scaleR-mtx-cnst-acc by (simp-all add: numeral-2-eq-2)

```

lemma *exp-mtx-cnst-acc-simps*:

```

exp (t *_R K) $$$ 1 $ 1 = 1 exp (t *_R K) $$$ 1 $ 2 = t exp (t *_R K) $$$ 1 $ 3 = t^2/2
exp (t *_R K) $$$ 2 $ 1 = 0 exp (t *_R K) $$$ 2 $ 2 = 1 exp (t *_R K) $$$ 2 $ 3 = t
exp (t *_R K) $$$ 3 $ 1 = 0 exp (t *_R K) $$$ 3 $ 2 = 0 exp (t *_R K) $$$ 3 $ 3 = 1
unfolding exp-mtx-cnst-acc one-mtx3 pow2-scaleR-mtx-cnst-acc by simp-all

```

lemma *exp-mtx-cnst-acc-vec-mult-eq*:  $\exp(t *_R K) *_V s =$

```

vector [s$$$ * t^2/2 + s$$$ * t + s$$$1, s$$$3 * t + s$$$2, s$$$3]
apply(subst exp-mtx-cnst-acc, subst pow2-scaleR-mtx-cnst-acc)
apply(simp add: sq-mtx-vec-mult-eq vector-def)
unfolding UNIV-3 by (simp add: fun-eq-iff)

```

**lemma** *local-flow-mtx-cnst-acc*:

*local-flow*  $((*_V) K)$  *UNIV UNIV*  $(\lambda t s. ((t *_R K)^2 /_R 2 + (t *_R K) + 1) *_V s)$   
**using** *local-flow-sq-mtx-linear*[of  $K$ ] **unfolding** *exp-mtx-cnst-acc* .

**lemma** *docking-station-arith*:

**assumes**  $(d::\text{real}) > x$  **and**  $v > 0$

**shows**  $(v = v^2 * t / (2 * d - 2 * x)) \longleftrightarrow (v * t - v^2 * t^2 / (4 * d - 4 * x) + x = d)$

**proof**

**assume**  $v = v^2 * t / (2 * d - 2 * x)$

**hence**  $v * t = 2 * (d - x)$

**using** *assms* **by** (*simp add: eq-divide-eq power2-eq-square*)

**hence**  $v * t - v^2 * t^2 / (4 * d - 4 * x) + x = 2 * (d - x) - 4 * (d - x)^2 / (4 * (d - x)) + x$

**apply**(*subst power-mult-distrib[symmetric]*)

**by** (*erule ssubst, subst power-mult-distrib, simp*)

**also have**  $\dots = d$

**apply**(*simp only: mult-divide-mult-cancel-left-if*)

**using** *assms* **by** (*auto simp: power2-eq-square*)

**finally show**  $v * t - v^2 * t^2 / (4 * d - 4 * x) + x = d$  .

**next**

**assume**  $v * t - v^2 * t^2 / (4 * d - 4 * x) + x = d$

**hence**  $0 = v^2 * t^2 / (4 * (d - x)) + (d - x) - v * t$

**by** *auto*

**hence**  $0 = (4 * (d - x)) * (v^2 * t^2 / (4 * (d - x)) + (d - x) - v * t)$

**by** *auto*

**also have**  $\dots = v^2 * t^2 + 4 * (d - x)^2 - (4 * (d - x)) * (v * t)$

**using** *assms* **apply**(*simp add: distrib-left right-diff-distrib*)

**apply**(*subst right-diff-distrib[symmetric]*) +

**by** (*simp add: power2-eq-square*)

**also have**  $\dots = (v * t - 2 * (d - x))^2$

**by** (*simp only: power2-diff, auto simp: field-simps power2-diff*)

**finally have**  $0 = (v * t - 2 * (d - x))^2$  .

**hence**  $v * t = 2 * (d - x)$

**by** *auto*

**thus**  $v = v^2 * t / (2 * d - 2 * x)$

**apply**(*subst power2-eq-square, subst mult.assoc*)

**apply**(*erule ssubst, subst right-diff-distrib[symmetric]*)

**using** *assms* **by** *auto*

**qed**

**lemma** *docking-station*:

**assumes**  $d > x_0$  **and**  $v_0 > 0$

**shows** *PRE*  $(\lambda s. s\$1 = x_0 \wedge s\$2 = v_0)$

*HP*  $((\exists :: (\lambda s. -(v_0^2 / (2 * (d - x_0))))); x' = (*_V) K \ \& \ G)$

*POST*  $(\lambda s. s\$2 = 0 \longleftrightarrow s\$1 = d)$

**apply**(*clarsimp simp: le-fun-def local-flow.fbox-g-ode-subset[OF local-flow-sq-mtx-linear[of  $K$ ]]*)

**unfolding** *exp-mtx-cnst-acc-vec-mult-eq* **using** *assms* **by** (*simp add: docking-station-arith*)

**no-notation** *mtx-cnst-acc* ( $K$ )

**end**

## 0.16 Examples

We prove partial correctness specifications of some hybrid systems with our verification components.

**theory** *ARCH2020-Examples*

**imports** *HS-VC-MKA-rel ../Matrices/MTX-Flows*

begin

### 0.16.1 Basic

**no-notation** *Archimedean-Field.ceiling* ( $\lceil \cdot \rceil$ )

#### Basic assignment

**lemma**  $\lceil \lambda s. s\$1 \geq (0::real) \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1)) \ \lceil \lambda s. s\$1 \geq 1 \rceil$   
**by** *simp*

#### Overwrite assignment on some branches

**lemma**  $\lceil \lambda s. s\$1 \geq (0::real) \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1))$   
 $(wp \ ((1 ::= (\lambda s. s\$1 + 1)) \cup (2 ::= (\lambda s. s\$1 + 1)))) \ \lceil \lambda s. s\$1 \geq 1 \rceil$   
**by** (*simp add: rel-aka.fbox-add2*)

#### Overwrite assignment in loop

**lemma**  $\lceil \lambda s. s\$1 \geq (0::real) \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1))$   
 $(wp \ (LOOP \ (1 ::= (\lambda s. s\$1 + 1)) \ INV \ (\lambda s. s\$1 \geq 1))) \ \lceil \lambda s. s\$1 \geq 1 \rceil$   
**apply**(*subst rel-aka.fbox-mult[symmetric]*)  
**by** (*rule wp-loopI-break, auto*)

#### Overwrite assignment in ODE

**lemma**  $0 \leq t \implies \lceil \lambda s. s\$1 \geq (0::real) \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1))$   
 $(wp \ (x' = (\lambda t s. (\chi \ i. \text{if } i=1 \text{ then } 2 \text{ else } 0)) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \ UNIV \ @ \ 0) \ \lceil \lambda s. s\$1 \geq 1 \rceil)$   
**apply**(*subst local-flow.wp-g-ode-ivl[where  $\varphi = \lambda t s. (\chi \ i. \text{if } i=1 \text{ then } 2*t+s\$1 \text{ else } s\$i)$  and  $T=UNIV$ ]*)  
**apply**(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff*)  
**apply**(*clarsimp, rule-tac x=1 in exI*) +  
**by** (*auto intro!: poly-derivatives*)

#### Overwrite with nondeterministic assignment

**lemma**  $\lceil \lambda s. s\$1 \geq (0::real) \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1)) \ (wp \ ((1 ::= ?); \lceil \lambda s. s\$1 \geq 1 \rceil) \ \lceil \lambda s. s\$1 \geq 1 \rceil)$   
**by** (*simp add: wp-rel, auto simp: p2r-def*)

#### Tests and universal quantification

**lemma**  $\lceil \lambda s::real^2. s\$1 \geq 0 \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1))$   
 $(wp \ ((\lceil \lambda s. s\$1 \geq 2 \rceil; (1 ::= (\lambda s. s\$1 - 1))) \cup (\lceil \lambda s. \forall i. s\$i \geq 1 \longrightarrow s\$2 \geq 1 \rceil; (1 ::= (\lambda s. s\$2))))$   
 $\lceil \lambda s. s\$1 \geq 1 \rceil$   
**by** (*simp add: wp-rel, auto simp: p2r-def assign-def vec-upd-def*)

#### Overwrite assignment several times

**lemma**  $0 \leq t \implies \lceil \lambda s::real^2. s\$1 \geq 0 \wedge s\$2 \geq 1 \rceil \leq wp \ (1 ::= (\lambda s. s\$1 + 1))$   
 $(wp \ ((LOOP \ (1 ::= (\lambda s. s\$1 + 1)) \ INV \ (\lambda s. s\$1 \geq 1)) \cup (2 ::= (\lambda s. s\$1 + 1))))$   
 $(wp \ (x' = (\lambda t s. (\chi \ i. \text{if } i=2 \text{ then } 2 \text{ else } 0)) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \ UNIV \ @ \ 0) \ (wp \ (1 ::= (\lambda s. s\$2))$   
 $\lceil \lambda s. s\$1 \geq 1 \rceil)))$   
**apply**(*simp, subst local-flow.wp-g-ode-ivl[where  $\varphi = \lambda t s. (\chi \ i. \text{if } i=2 \text{ then } 2*t+s\$2 \text{ else } s\$i)$  and  $T=UNIV$ ]*)  
**apply**(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff*)  
**apply** (*clarsimp, rule-tac x=1 in exI, force*)  
**apply**(*auto intro!: poly-derivatives vec-eq-iff*)[1]  
**apply**(*subst change-loopI[where  $I = \lambda s. 1 \leq s\$1 \wedge 1 \leq s\$2$ ]*)  
**apply**(*subst rel-aka.fbox-mult[symmetric]*)  
**apply**(*rule rel-aka.fbox-seq-var*)  
**apply**(*subst wp-assign[where  $Q = \lambda s. 1 \leq s\$1 \wedge 1 \leq s\$2$ ], simp*)  
**apply**(*subst le-wp-choice-iff, rule conjI*)  
**by** (*subst wp-loopI, auto*)

### Potentially overwrite dynamics

**lemma**  $0 \leq t \implies [\lambda s :: \text{real}^2. s\$1 > 0 \wedge s\$2 > 0] \leq$   
 $\text{wp } (x' = (\lambda t s. (\chi i. \text{if } i=1 \text{ then } 5 \text{ else } 0))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \text{ UNIV @ } 0)$   
 $(\text{wp } ((\text{LOOP } (1 ::= (\lambda s. s\$1 + 3))) \text{ INV } (\lambda s. 0 < s\$1))) \cup (2 ::= (\lambda s. s\$1)))$   
 $[\lambda s. s\$1 > 0 \wedge s\$2 > 0])$   
**apply**(subst rel-aka.fbox-mult[symmetric])+  
**apply**(rule rel-aka.fbox-seq-var)+  
**apply**(subst local-flow.wp-g-ode-ivl[**where**  $\varphi = \lambda t s. (\chi i. \text{if } i=1 \text{ then } 5*t + s\$1 \text{ else } s\$i)$   
**and**  $T = \text{UNIV}$  **and**  $Q = \lambda s. s\$1 > 0 \wedge s\$2 > 0$ ]; simp?)+  
**apply**(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)+  
**apply**(clarsimp, rule-tac x=1 **in** exI)+  
**apply**(force, force intro!: poly-derivatives)+  
**apply**(subst le-wp-choice-iff, rule conjI)+  
**apply**(subst change-loopI[**where**  $I = \lambda s. s\$1 > 0 \wedge s\$2 > 0$ ])  
**by** (rule wp-loopI, simp-all)

### Potentially overwrite exponential decay

**abbreviation**  $\text{po-exp-dec-f} :: \text{real}^2 \Rightarrow \text{real}^2 (f)$   
**where**  $f s \equiv (\chi i. \text{if } i=1 \text{ then } -s\$1 \text{ else } 0)$

**abbreviation**  $\text{po-exp-dec-flow} :: \text{real} \Rightarrow \text{real}^2 \Rightarrow \text{real}^2 (\varphi)$   
**where**  $\varphi t s \equiv (\chi i. \text{if } i=1 \text{ then } s\$1 * \exp(-t) \text{ else } s\$i)$

**lemma**  $\text{local-flow-exp-flow: local-flow } f \text{ UNIV UNIV } \varphi$   
**apply**(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def)  
**apply**(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 **in** exI)+  
**apply**(unfold UNIV-2, simp)+  
**apply**(metis power2-commute real-sqrt-ge-abs1)+  
**by** (auto intro!: poly-derivatives simp: forall-2 vec-eq-iff)

**lemma**  $0 \leq t \implies [\lambda s :: \text{real}^2. s\$1 > 0 \wedge s\$2 > 0] \leq \text{wp } (x' = (\lambda t. f)) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \text{ UNIV @ } 0)$   
 $(\text{wp } ((\text{LOOP } (1 ::= (\lambda s. s\$1 + 3))) \text{ INV } (\lambda s. 0 < s\$1))) \cup (2 ::= (\lambda s. s\$1)))$   
 $[\lambda s. s\$1 > 0 \wedge s\$2 > 0])$   
**apply**(subst rel-aka.fbox-mult[symmetric])+  
**apply**(rule rel-aka.fbox-seq-var)+  
**apply**(subst local-flow.wp-g-ode-ivl[ $\text{OF local-flow-exp-flow}$ , **where**  $Q = \lambda s. s\$1 > 0 \wedge s\$2 > 0$ ]; simp)+  
**apply**(subst le-wp-choice-iff, rule conjI)+  
**apply**(subst change-loopI[**where**  $I = \lambda s. s\$1 > 0 \wedge s\$2 > 0$ ])  
**by** (rule wp-loopI, auto)

**no-notation**  $\text{po-exp-dec-f } (f)$   
**and**  $\text{po-exp-dec-flow } (\varphi)$

### Dynamics: Cascaded

**lemma**  $0 \leq t \implies [\lambda s :: \text{real}^1. s\$1 > 0] \leq$   
 $\text{wp } ((x' = (\lambda t s. (\chi i. \text{if } i=1 \text{ then } 5 \text{ else } 0))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \text{ UNIV @ } 0);$   
 $(x' = (\lambda t s. (\chi i. \text{if } i=1 \text{ then } 2 \text{ else } 0))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \text{ UNIV @ } 0);$   
 $(x' = (\lambda t s. (\chi i. \text{if } i=1 \text{ then } s\$1 \text{ else } 0))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \text{ UNIV @ } 0)$   
 $[\lambda s. s\$1 > 0]$   
**apply**(simp, subst local-flow.wp-g-ode-ivl[**where**  $T = \text{UNIV}$  **and**  $\varphi = \lambda t s. (\chi i. s\$1 * \exp t)$ ]; simp?)+  
**apply**(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)+  
**apply**(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 **in** exI)+  
**apply**(force, force intro!: poly-derivatives)+  
**apply**(subst local-flow.wp-g-ode-ivl[**where**  $T = \text{UNIV}$  **and**  $\varphi = \lambda t s. (\chi i. 2*t + s\$1)$ ]; simp?)+  
**apply**(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)+  
**apply**(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 **in** exI)+

```

  apply(force, force intro!: poly-derivatives)
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and  $\varphi=\lambda t s. (\chi i. 5*t+s\$1)$ ]; simp?)
  apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  apply(force, force intro!: poly-derivatives)
  by auto (smt exp-gt-zero mult-minus-left real-mult-less-iff1)

```

### Dynamics: Single integrator time

```

lemma 0 ≤ t ⇒ [λs::real^1. s$1 = 0] ≤ wp (x'=(λt s. (χ i. 1)) & G on (λs. {0..t}) UNIV @ 0) [λs.
s$1 ≥ 0]
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and  $\varphi=\lambda t s. (\chi i. t+s\$1)$ ]; simp?)
  apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  by (auto intro!: poly-derivatives)

```

### Dynamics: Single integrator

```

lemma 0 ≤ t ⇒ [λs::real^2. s$1 ≥ 0 ∧ s$2 ≥ 0] ≤
  wp (x'=(λt s. (χ i. if i = 1 then s$2 else 0)) & G on (λs. {0..t}) UNIV @ 0) [λs. s$1 ≥ 0]
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and
     $\varphi=\lambda t s. (\chi i. if i = 1 then s\$2*t+s\$1 else s\$i)$ ]; simp?)
  apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
  apply(simp add: dist-norm norm-vec-def L2-set-def)
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  unfolding UNIV-2 by (auto intro!: poly-derivatives simp: forall-2 vec-eq-iff)

```

### Dynamics: Double integrator

```

lemma 0 ≤ t ⇒ [λs::real^3. s$1 ≥ 0 ∧ s$2 ≥ 0 ∧ s$3 ≥ 0] ≤
  wp (x'=(λt s. (χ i. if i = 1 then s$2 else (if i = 2 then s$3 else 0))) & G on (λs. {0..t}) UNIV @ 0)
  [λs. s$1 ≥ 0]
  apply(subst local-flow.wp-g-ode-ivl[where T=UNIV and
     $\varphi=\lambda t s. (\chi i. if i = 1 then s\$3*t^2/2 + s\$2*t+s\$1 else (if i = 2 then s\$3*t+s\$2 else s\$i))$ ]; simp?)
  apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI)+
  unfolding UNIV-3 by (auto intro!: poly-derivatives simp: forall-3 vec-eq-iff)

```

### Dynamics: Triple integrator

```

abbreviation triple-int-f :: real^4 ⇒ real^4 (f)
  where f s ≡ (χ i. if i = 1 then s$2 else (if i = 2 then s$3 else (if i = 3 then s$4 else (s$4)^2)))

```

```

lemma 0 ≤ t ⇒ [λs::real^4. s$1 ≥ 0 ∧ s$2 ≥ 0 ∧ s$3 ≥ 0 ∧ s$4 ≥ 0] ≤
  wp (x'=(λt. f) & G on (λs. {0..t}) UNIV @ 0)
  [λs. s$1 ≥ 0]
  apply(rule-tac C=λs. s$4 ≥ 0 in diff-cut-rule)
  apply(subst g-ode-inv-def[symmetric, where I=λs. s$4 ≥ 0])
  apply(rule wp-g-odei; simp?)
  apply(rule-tac ν'=λs. 0 and μ'=λs. (s$4)^2 in diff-invariant-leq-rule; simp?)
  apply(force intro!: poly-derivatives simp: forall-4)
  apply(rule-tac C=λs. s$3 ≥ 0 in diff-cut-rule, simp-all)
  apply(subst g-ode-inv-def[symmetric, where I=λs. s$3 ≥ 0])
  apply(rule wp-g-odei; simp?)
  apply(rule-tac ν'=λs. 0 and μ'=λs. (s$4) in diff-invariant-rules(2); simp?)
  apply(force intro!: poly-derivatives simp: forall-4)
  apply(rule-tac C=λs. s$2 ≥ 0 in diff-cut-rule, simp-all)
  apply(subst g-ode-inv-def[symmetric, where I=λs. s$2 ≥ 0])
  apply(rule wp-g-odei; simp?)

```

```

apply(rule-tac  $\nu' = \lambda s. 0$  and  $\mu' = \lambda s. (s\$3)$  in diff-invariant-rules(2); simp?)
apply(force intro!: poly-derivatives simp: forall-4)
apply(rule-tac  $C = \lambda s. s\$1 \geq 0$  in diff-cut-rule, simp-all)
apply(subst g-ode-inv-def[symmetric, where  $I = \lambda s. s\$1 \geq 0$ ])
apply(rule wp-g-odei; simp?)
apply(rule-tac  $\nu' = \lambda s. 0$  and  $\mu' = \lambda s. (s\$2)$  in diff-invariant-rules(2); simp?)
apply(force intro!: poly-derivatives simp: forall-4)
by (rule diff-weak-rule, simp)

```

**no-notation** triple-int-f (f)

### Dynamics: Exponential decay (1)

```

lemma  $0 \leq t \implies [\lambda s::\text{real}^1. s\$1 > 0] \leq wp (x' = (\lambda s. (\chi i. - s\$1))) \ \& \ G \text{ on } [\lambda s. s\$1 > 0]$ 
apply(subst local-flow.wp-g-ode-subset[where  $T = UNIV$  and  $\varphi = \lambda t s. (\chi i. s\$1 * \exp(-t))$ ]; simp?)
apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac  $x=1$  in exI)+
by (auto intro!: poly-derivatives)

```

### Dynamics: Exponential decay (2)

```

lemma  $0 \leq t \implies [\lambda s::\text{real}^1. s\$1 > 0] \leq wp (x' = (\lambda t s. (\chi i. - s\$1 + 1))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \ UNIV$ 
 $\text{@ } 0) [\lambda s. s\$1 > 0]$ 
apply(subst local-flow.wp-g-ode-ivl[where  $T = UNIV$  and  $\varphi = \lambda t s. (\chi i. 1 - \exp(-t) + s\$1 * \exp(-t))$ ];clarsimp?)
apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac  $x=1$  in exI)+
by (auto intro!: poly-derivatives simp: field-simps) (smt exp-gt-zero mult-pos-pos one-less-exp-iff)

```

### Dynamics: Exponential decay (3)

```

lemma  $0 \leq t \implies y > 0 \implies [\lambda s::\text{real}^1. s\$1 > 0] \leq$ 
 $wp (x' = (\lambda t s. (\chi i. - y * s\$1))) \ \& \ G \text{ on } (\lambda s. UNIV) \ UNIV \text{ @ } 0) [\lambda s. s\$1 > 0]$ 
apply(subst local-flow.wp-g-ode[where  $T = UNIV$  and  $\varphi = \lambda t s. (\chi i. s\$1 * (\exp(-t * y)))$ ];clarsimp?)
apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
apply(clarsimp, rule-tac  $x=1$  in exI)
apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac  $x=y$  in exI)
apply (metis abs-mult abs-of-pos dist-commute dist-real-def less-eq-real-def
vector-space-over-itself.scale-right-diff-distrib)
by (auto intro!: poly-derivatives simp: field-simps)

```

### Dynamics: Exponential growth (1)

```

lemma  $0 \leq t \implies [\lambda s::\text{real}^1. s\$1 \geq 0] \leq$ 
 $wp (x' = (\lambda t s. (\chi i. s\$1))) \ \& \ G \text{ on } (\lambda s. \{0..t\}) \ UNIV \text{ @ } 0) [\lambda s. s\$1 \geq 0]$ 
apply(subst local-flow.wp-g-ode-ivl[where  $T = UNIV$  and  $\varphi = \lambda t s. (\chi i. s\$1 * \exp t)$ ];clarsimp?)
apply(unfold-locales; (simp add: local-lipschitz-def lipschitz-on-def vec-eq-iff)?)
apply(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac  $x=1$  in exI)+
by (auto intro!: poly-derivatives)

```

### Dynamics: Exponential growth (2)

```

lemma  $[\lambda s::\text{real}^2. s\$1 \geq 0 \wedge s\$2 \geq 0] \leq$ 
 $wp (x' = (\lambda t s. (\chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (s\$2)^2))) \ \& \ G \text{ on } (\lambda s. \{0..\}) \ \{s. s\$2 < 0\} \text{ @ } 0) [\lambda s. s\$1 \geq 0]$ 
apply(rule-tac  $C = \lambda s. s\$2 \geq 0$  in diff-cut-rule)
apply(subst g-ode-inv-def[symmetric, where  $I = \lambda s. s\$2 \geq 0$ ], rule wp-g-odei; simp?)
apply(rule-tac  $\nu' = \lambda s. 0$  and  $\mu' = \lambda s. (s\$2)^2$  in diff-invariant-rules(2); (simp add: forall-2)?)
apply(rule-tac  $C = \lambda s. s\$1 \geq 0$  in diff-cut-rule, simp-all)
apply(subst g-ode-inv-def[symmetric, where  $I = \lambda s. s\$1 \geq 0$ ])
apply(rule wp-g-odei; simp?)
apply(rule-tac  $\nu' = \lambda s. 0$  and  $\mu' = \lambda s. (s\$2)$  in diff-invariant-rules(2); (simp add: forall-2)?)

```



by (rule *diff-weak-rule*, *simp*)

### Dynamics: Exponential growth (4)

**lemma**  $0 \leq t \implies [\lambda s::\text{real}^1. s\$1 > 0] \leq$   
 $wp \ (x' = (\lambda t \ s. (\chi \ i. (s\$1)^2)) \ \& \ G \ \text{on} \ (\lambda s. \{0..t\}) \ UNIV \ @ \ 0) \ [\lambda s. s\$1 > 0]$   
**apply**(subst *g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$1 > 0$ ], rule *wp-g-odei*, *simp-all*)  
**by** (rule-tac  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. (s\$1)^2$  **in** *diff-invariant-rules*(3); *simp?*)

### Dynamics: Exponential growth (5)

**lemma**  $0 \leq t \implies [\lambda s::\text{real}^1. s\$1 \geq 1] \leq$   
 $wp \ (x' = (\lambda t \ s. (\chi \ i. (s\$1)^2 + 2 \cdot (s\$1)^4)) \ \& \ G \ \text{on} \ (\lambda s. \{0..t\}) \ UNIV \ @ \ 0) \ [\lambda s. s\$1^3 \geq (s\$1)^2]$   
**apply**(rule-tac  $C = \lambda s. s\$1 \geq 1$  **in** *diff-cut-rule*; *simp?*)  
**apply** (rule-tac  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. (s\$1)^2 + 2 \cdot s\$1^4$  **in** *diff-invariant-rules*(2); *simp?*)  
**apply**(force *intro!*: *poly-derivatives*)  
**apply**(subst *g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$1^3 \geq (s\$1)^2$ ], rule *wp-g-odei*, *simp-all* add:  
*power-increasing*)  
**apply** (rule-tac  $\nu' = \lambda s. 2 \cdot s\$1 \cdot ((s\$1)^2 + 2 \cdot (s\$1)^4)$   
**and**  $\mu' = \lambda s. 3 \cdot (s\$1)^2 \cdot ((s\$1)^2 + 2 \cdot (s\$1)^4)$  **in** *diff-invariant-rules*(2); *clarsimp?*)  
**apply**(auto *intro!*: *poly-derivatives* *simp*: *field-simps* *semiring-normalization-rules*(27,28))  
**apply**(subgoal-tac  $X \ \tau \ \$1 \geq 1$ )  
**apply**(subgoal-tac  $2 + 4 \cdot (X \ \tau) \$1^2 \leq 3 \cdot (X \ \tau) \$1 + 6 \cdot (X \ \tau) \$1^3$ )  
**apply** (smt *One-nat-def* *numerals*(1) *one-le-power* *power.simps*(2) *power-0* *power-add-numeral* *power-less-power-Suc*  
*power-one-right* *semiring-norm*(2) *semiring-norm*(4) *semiring-norm*(5))  
**apply** (smt *numeral-One* *one-le-power* *power-add-numeral* *power-commutes* *power-le-one* *power-less-power-Suc*  
*power-one-right* *semiring-norm*(5))  
**by** *simp*

### Dynamics: Rotational dynamics (1)

**lemma**  $[\lambda s::\text{real}^2. (s\$1)^2 + (s\$2)^2 = 1] \leq wp \ (x' = (\lambda s. (\chi \ i. \text{if } i = 1 \text{ then } -s\$2 \text{ else } s\$1)) \ \& \ G)$   
 $[\lambda s. (s\$1)^2 + (s\$2)^2 = 1]$   
**by** (auto *intro!*: *poly-derivatives* *diff-invariant-rules*)

### Dynamics: Rotational dynamics (2)

**abbreviation** *rot-dyn2-mtx* ::  $3 \text{ sq-mtx } (A)$   
**where**  $A \equiv \text{mtx}$   
 $([0, -1, 0] \#$   
 $[0, 0, 1] \#$   
 $[0, -1, 0] \# [])$

**abbreviation** *rot-dyn2-f* ::  $\text{real}^3 \Rightarrow \text{real}^3 \ (f)$   
**where**  $f \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } -s\$2 \text{ else } (\text{if } i = 2 \text{ then } s\$3 \text{ else } -s\$2))$

**lemma** *rot-dyn2-f-linear*:  $f \ s = A *_{\text{V}} s$   
**apply**(*simp* add: *vec-eq-iff* *sq-mtx-vec-mult-eq*)  
**unfolding** *UNIV-3* **using** *exhaust-3* **by** *force*

**abbreviation** *rot-dyn2-flow* ::  $\text{real} \Rightarrow \text{real}^3 \Rightarrow \text{real}^3 \ (\varphi)$   
**where**  $\varphi \ t \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } -s\$3 + s\$1 + s\$3 * \cos t - s\$2 * \sin t$   
 $\text{else } (\text{if } i = 2 \text{ then } s\$3 * \sin t + s\$2 * \cos t \text{ else } s\$3 * \cos t - s\$2 * \sin t))$

**lemma** *mtx-circ-flow-eq*:  $\exp (t *_{\text{R}} A) *_{\text{V}} s = \varphi \ t \ s$   
**apply**(rule *local-flow.eq-solution*[*OF* *local-flow-sq-mtx-linear*, *symmetric*, **where**  $U = \lambda s. UNIV$ ])  
**apply**(*simp-all*, rule *ivp-solsI*, *simp-all* add: *sq-mtx-vec-mult-eq* *vec-eq-iff*)  
**unfolding** *UNIV-3* **using** *exhaust-3*  
**by** (force *intro!*: *poly-derivatives* *simp*: *matrix-vector-mult-def*)+

**lemma**  $\lceil \lambda s :: \text{real}^3. (s\$1)^2 + (s\$2)^2 = 1 \wedge s\$3 = s\$1 \rceil \leq \text{wp } (x' = f \ \& \ G)$   
 $\lceil \lambda s. (s\$1)^2 + (s\$2)^2 = 1 \wedge s\$3 = s\$1 \rceil$   
**apply**(*subst rot-dyn2-f-linear*, *subst local-flow.wp-g-ode-subset[OF local-flow-sq-mtx-linear]*)  
**unfolding** *mtx-circ-flow-eq* **by** *auto*

**no-notation** *rot-dyn2-f* (*f*)  
**and** *rot-dyn2-mtx* (*A*)  
**and** *rot-dyn2-flow* ( $\varphi$ )

### Dynamics: Rotational dynamics (3)

**lemma**  $\lceil \lambda s :: \text{real}^4. (s\$3)^2 + (s\$4)^2 = w^2 \cdot p^2 \wedge s\$3 = -w \cdot s\$2 \wedge s\$4 = w \cdot s\$1 \rceil \leq$   
 $\text{wp } (x' = (\lambda s. (\chi \ i. \text{if } i=1 \text{ then } s\$3 \text{ else } (\text{if } i=2 \text{ then } s\$4 \text{ else } (\text{if } i=3 \text{ then } -w * s\$4 \text{ else } w * s\$3)))))) \ \& \ G)$   
 $\lceil \lambda s. (s\$3)^2 + (s\$4)^2 = w^2 * p^2 \wedge s\$3 = -w * s\$2 \wedge s\$4 = w * s\$1 \rceil$   
**by** (*auto intro! diff-invariant-rules poly-derivatives*)

### Dynamics: Spiral to equilibrium

**lemma**  $\lceil \lambda s :: \text{real}^3. (s\$3) \geq 0 \wedge s\$1=0 \wedge s\$2=3 \rceil \leq$   
 $\text{wp } (x' = (\lambda t \ s. (\chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } - (s\$3)^2 * (s\$1) - 2 * (s\$3) * (s\$2) \text{ else } 0))) \ \& \ G \text{ on } (\lambda s. \{0..\}) \text{ UNIV @ } 0)$   
 $\lceil \lambda s. (s\$3)^2 * (s\$1)^2 + (s\$2)^2 \leq 9 \rceil$   
**apply**(*rule-tac C = λs. s\$3 ≥ 0 in diff-cut-rule; simp?*)  
**apply**(*subst g-ode-inv-def[symmetric, where I = λs. s\$3 ≥ 0], rule wp-g-odei, simp-all*)  
**apply**(*rule-tac ν' = λs. 0 and μ' = λs. 0 in diff-invariant-rules(2); (simp add: forall-3)?*)  
**apply**(*subst g-ode-inv-def[symmetric, where I = λs. (s\$3)^2 \* (s\$1)^2 + (s\$2)^2 ≤ 9], rule wp-g-odei, simp-all*  
*add: power-increasing*)  
**apply**(*rule-tac ν' = λs. 2 \* (s\$3)^2 \* (s\$1) \* (s\$2) + 2 \* (s\$2) \* (-(s\$3)^2 \* (s\$1) - 2 \* (s\$3) \* (s\$2)) and*  
 $\mu' = \lambda s. 0$  **in** *diff-invariant-rules(2)*)  
**by** (*auto intro! poly-derivatives simp: forall-3 field-simps*) (*simp add: mult.assoc[symmetric] power2-eq-square*)

### Dynamics: Open cases

**lemma** *has-vderiv-mono-test*:  
**assumes** *T-hyp*: *is-interval T*  
**and** *d-hyp*:  $D f = f'$  *on T*  
**and** *xy-hyp*:  $x \in T \ y \in T \ x \leq y$   
**shows**  $\forall x \in T. (0 :: \text{real}) \leq f' x \implies f x \leq f y$   
**and**  $\forall x \in T. f' x \leq 0 \implies f x \geq f y$   
**proof**–  
**have**  $\{x..y\} \subseteq T$   
**using** *T-hyp xy-hyp* **by** (*meson atLeastAtMost-iff mem-is-interval-1-I subsetI*)  
**hence**  $D f = f'$  *on*  $\{x..y\}$   
**using** *has-vderiv-on-subset[OF d-hyp(1)]* **by** *blast*  
**hence**  $(\bigwedge t. x \leq t \implies t \leq y \implies D f \mapsto (\lambda \tau. \tau *_R f' t) \text{ at } t \text{ within } \{x..y\})$   
**unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *auto*  
**then obtain** *c* **where** *c-hyp*:  $c \in \{x..y\} \wedge f y - f x = (y - x) *_R f' c$   
**using** *mvt-very-simple[OF xy-hyp(3), of f (λt τ. τ \*\_R f' t)]* **by** *blast*  
**hence** *mvt-hyp*:  $f x = f y - f' c * (y - x)$   
**by** (*simp add: mult.commute*)  
**also have**  $\forall x \in T. 0 \leq f' x \implies \dots \leq f y$   
**using** *xy-hyp d-hyp c-hyp*  $\langle \{x..y\} \subseteq T \rangle$  **by** *auto*  
**finally show**  $\forall x \in T. 0 \leq f' x \implies f x \leq f y$ .  
**have**  $\forall x \in T. f' x \leq 0 \implies f y - f' c * (y - x) \geq f y$   
**using** *xy-hyp d-hyp c-hyp*  $\langle \{x..y\} \subseteq T \rangle$  **by** (*auto simp: mult-le-0-iff*)  
**thus**  $\forall x \in T. f' x \leq 0 \implies f x \geq f y$   
**using** *mvt-hyp* **by** *auto*  
**qed**

**lemma** *continuous-on-ge-ball-ge*:

*continuous-on*  $T f \implies x \in T \implies f x > (k::real) \implies \exists \varepsilon > 0. \forall y \in \text{ball } x \ \varepsilon \cap T. f y > k$   
**unfolding** *continuous-on-iff* **apply**(*erule-tac*  $x=x$  **in** *ballE*; *clarsimp?*)  
**apply**(*erule-tac*  $x=f x - k$  **in** *allE*, *clarsimp simp: dist-norm*)  
**apply**(*rename-tac*  $\delta$ , *erule-tac*  $x=\delta$  **in** *exI*, *clarsimp*)  
**apply**(*erule-tac*  $x=y$  **in** *ballE*; *clarsimp?*)  
**by** (*subst (asm) abs-le-eq*, *simp-all add: dist-commute*)

**lemma** *current-vderiv-ge-always-ge*:

**fixes**  $c::real$   
**assumes** *init*:  $c < x \ t_0$  **and** *ode*:  $D \ x = x'$  *on*  $\{t_0..\}$   
**and** *dhyp*:  $x' = (\lambda t. g \ (x \ t)) \ \forall x \geq c. g \ x \geq 0$   
**shows**  $\forall t \geq t_0. x \ t > c$

**proof**—

**have** *cont*: *continuous-on*  $\{t_0..\}$   $x$   
**using** *vderiv-on-continuous-on*[*OF ode*] .  
**{assume**  $\exists t \geq t_0. x \ t \leq c$   
**hence** *inf*:  $\{t. t > t_0 \wedge x \ t \leq c\} \neq \{\}$  *bdd-below*  $\{t. t > t_0 \wedge x \ t \leq c\}$   
**using** *init less-eq-real-def* **unfolding** *bdd-below-def* **by** *force* (*rule-tac*  $x=t_0$  **in** *exI*, *simp*)  
**define**  $t_1$  **where** *t1-hyp*:  $t_1 = \text{Inf } \{t. t > t_0 \wedge x \ t \leq c\}$   
**hence**  $t_0 \leq t_1$   
**using** *le-cInf-iff*[*OF inf*, *of t0*] **by** *auto*  
**have**  $x \ t_1 \geq c$   
**proof**—  
**{assume**  $x \ t_1 < c$   
**hence** *obs*:  $x \ t_1 \leq c \ x \ t_0 \geq c \ t_1 \neq t_0$   
**using** *init* **by** *auto*  
**hence**  $t_1 > t_0$   
**using**  $\langle t_0 \leq t_1 \rangle$  **by** *auto*  
**then obtain**  $k$  **where** *k2-hyp*:  $k \geq t_0 \wedge k \leq t_1 \wedge x \ k = c$   
**using** *IVT2'*[*of*  $\lambda t. x \ t$ , *OF obs*(1,2) - *continuous-on-subset*[*OF cont*]] **by** *auto*  
**hence**  $t_0 < k \ k < t_1$   
**using**  $\langle x \ t_1 < c \rangle$  *less-eq-real-def* *init* **by** *auto*  
**also have**  $t_1 \leq k$   
**using** *cInf-lower*[*OF - inf*(2)] *k2-hyp* *calculation* **unfolding** *t1-hyp* **by** *auto*  
**ultimately have** *False*  
**by** *simp* }  
**thus**  $x \ t_1 \geq c$   
**by** *fastforce*

**qed**

**hence** *obs*:  $\forall t \in \{t_0..<t_1\}. x \ t > c$

**proof**—

**{assume**  $\exists t \in \{t_0..<t_1\}. x \ t \leq c$   
**then obtain**  $k$  **where** *k2-hyp*:  $k \geq t_0 \wedge k < t_1 \wedge x \ k \leq c$   
**by** *auto*  
**hence**  $k > t_0$   
**using**  $\langle x \ t_0 > c \rangle$  *less-eq-real-def* **by** *auto*  
**hence**  $t_1 \leq k$   
**using** *cInf-lower*[*OF - inf*(2)] *k2-hyp* **unfolding** *t1-hyp* **by** *auto*  
**hence** *False*  
**using** *k2-hyp* **by** *auto* }  
**thus**  $\forall t \in \{t_0..<t_1\}. x \ t > c$   
**by** *force*

**qed**

**hence**  $\forall t \in \{t_0..t_1\}. x' \ t \geq 0$

**using**  $\langle x \ t_1 \geq c \rangle$  *dhyp*(2) *less-eq-real-def*  
**unfolding** *dhyp* **by** (*metis atLeastAtMost-iff atLeastLessThan-iff*)

**hence**  $x \ t_0 \leq x \ t_1$

**apply**(*rule-tac*  $f=\lambda t. x \ t$  **and**  $T=\{t_0..t_1\}$  **in** *has-vderiv-mono-test*(1); *clarsimp*)

```

using has-vderiv-on-subset[OF ode] apply force
using ⟨t0 ≤ t1⟩ by (auto simp: less-eq-real-def)
hence c < x t1
using ⟨x t1 ≥ c⟩ init by auto
then obtain ε where eps-hyp: ε > 0 ∧ (∀ t ∈ ball t1 ε ∩ {t0..}. c < x t)
using continuous-on-ge-ball-ge[of - λt. x t, OF cont - ⟨c < x t1⟩] ⟨t0 ≤ t1⟩ by auto
hence ∀ t ∈ {t0..<t1+ε}. c < x t
using obs ⟨t0 ≤ t1⟩ ball-eq-greaterThanLessThan by auto
hence ∀ t ∈ {t. t > t0 ∧ x t ≤ c}. t1+ε ≤ t
by (metis (mono-tags, lifting) atLeastLessThan-iff less-eq-real-def mem-Collect-eq not-le)
hence t1+ε ≤ t1
using le-cInf-iff[OF inf] unfolding t1-hyp by simp
hence False
using eps-hyp by auto}
thus ∀ t ≥ t0. c < x t
by fastforce
qed

```

```

lemma 0 ≤ t ⟹ [λs::real^2. s$1^3 > 5 ∧ s$2 > 2] ≤
wp (x' = (λt s. (χ i. if i=1 then s$1^3 + s$1^4 else 5 * s$2 + s$2^2)) & G on (λs. {0..}) UNIV @ 0)
[λs. s$1^3 > 5 ∧ s$2 > 2]
apply(simp, rule diff-invariant-rules, simp-all add: diff-invariant-eq ivp-sols-def forall-2; clarsimp)
apply(frule-tac x=λt. X t $ 1 ^ 3 and g=λt. 3 * t^2 + 3 * (root 3 t)^5 in current-vderiv-ge-always-ge)
apply(rule poly-derivatives, simp, assumption, simp)
apply(force simp: field-simps odd-real-root-power-cancel, force simp: add-nonneg-pos, force)
apply(frule-tac x=λt. X t $ 2 in current-vderiv-ge-always-ge)
by (force, force, force simp: add-nonneg-pos, simp)

```

### Dynamics: Closed cases

```

lemma z = - 2 ⟹ [λs::real^2. s$1 ≥ 1 ∧ s$2 = 10] ≤
wp (x' = (λs. (χ i. if i=1 then s$2 else z + (s$2)^2 - s$2)) & (λs. s$2 ≥ 0))
[λs. s$1 ≥ 1 ∧ s$2 ≥ 0]
apply(subst g-ode-inv-def[symmetric, where I=λs. s$1 ≥ 1 ∧ s$2 ≥ 0], rule wp-g-odei, simp-all)
apply(rule diff-invariant-rules)
apply(rule-tac ν'=λs. 0 and μ'=λs. s$2 in diff-invariant-rules(2), simp-all add: diff-invariant-eq)
by (force intro!: poly-derivatives)

```

### Dynamics: Conserved quantity

```

lemma dyn-cons-qty-arith: (36::real) · (x1^2 · (x1 · x2 ^ 3)) -
(- (24 · (x1^2 · x2) · x1 ^ 3 · (x2)^2) - 12 · (x1^2 · x2) · x1 · x2 ^ 4) -
36 · (x1^2 · (x2 · x1)) · (x2)^2 - 12 · (x1^2 · (x1 · x2 ^ 5)) = 24 · (x1 ^ 5 · x2 ^ 3)
(is ?t1 - (- ?t2 - ?t3) - ?t4 - ?t5 = ?t6)

```

proof—

```

have eq1: ?t1 = ?t4
by (simp add: power2-eq-square power3-eq-cube)
have eq2: - (- ?t2 - ?t3) = (?t6 + ?t3)
by (auto simp: field-simps semiring-normalization-rules(27))
have eq3: ?t3 = ?t5
by (auto simp: field-simps semiring-normalization-rules(27))
show ?t1 - (- ?t2 - ?t3) - ?t4 - ?t5 = ?t6
unfolding eq1 eq2 eq3 by (simp add: field-simps semiring-normalization-rules(27))

```

qed

```

lemma 0 ≤ t ⟹ [λs::real^2. (s$1)^4*(s$2)^2+(s$1)^2*(s$2)^4-3*(s$1)^2*(s$2)^2 + 1 ≤ c] ≤
wp (x' = (λt s. (χ i. if i=1 then 2*(s$1)^4*(s$2)+4*(s$1)^2*(s$2)^3-6*(s$1)^2*(s$2)
else -4*(s$1)^3*(s$2)^2-2*(s$1)*(s$2)^4+6*(s$1)*(s$2)^2)) & G on (λs. {0..}) UNIV @ 0)

```

```

[ $\lambda s. (s\$1)^4 * (s\$2)^2 + (s\$1)^2 * (s\$2)^4 - 3 * (s\$1)^2 * (s\$2)^2 + 1 \leq c$ ]
apply(simp, rule-tac  $\mu' = \lambda s. 0$  and  $\nu' = \lambda s. 0$  in diff-invariant-rules(2); clarsimp simp: forall-2)
apply(intro poly-derivatives; (assumption)?, (rule poly-derivatives)?)
apply force+
apply(clarsimp simp: algebra-simps(17,18,19,20) semiring-normalization-rules(27,28))
by (auto simp: dyn-cons-qty-arith)

```

### Dynamics: Darboux equality

**lemma** *mult-abs-right-mono*:  $a < b \implies a * |c| \leq b * |c|$  **for**  $c::real$   
**by** (simp add: mult-right-mono)

**lemma** *local-lipschitz-first-order-linear*:

**fixes**  $c::real \Rightarrow real$

**assumes** continuous-on  $T\ c$

**shows** local-lipschitz  $T\ UNIV\ (\lambda t. (*)\ (c\ t))$

**proof**(unfold local-lipschitz-def lipschitz-on-def, clarsimp simp: dist-norm)

**fix**  $x\ t::real$  **assume**  $t \in T$

**then obtain**  $\delta$  **where**  $d\text{-hyp}$ :  $\delta > 0 \wedge (\forall \tau \in T. |\tau - t| < \delta \longrightarrow |c\ \tau - c\ t| < \max 1\ |c\ t|)$

**using** *assms* **unfolding** continuous-on-iff

**apply**(erule-tac  $x=t$  **in** ballE, erule-tac  $x=\max 1\ (|c\ t|)$  **in** allE; clarsimp)

**by** (metis dist-norm less-max-iff-disj real-norm-def zero-less-one)

**{fix**  $\tau\ x_1\ x_2$

**assume**  $\tau \in cball\ t\ (\delta/2) \cap T\ x_1 \in cball\ x\ (\delta/2)\ x_2 \in cball\ x\ (\delta/2)$

**hence**  $|\tau - t| < \delta\ \tau \in T$

**by** (auto simp: dist-norm, smt d-hyp)

**hence**  $|c\ \tau - c\ t| < \max 1\ |c\ t|$

**using** d-hyp **by** auto

**hence**  $-(\max 1\ |c\ t| + |c\ t|) < c\ \tau \wedge c\ \tau < \max 1\ |c\ t| + |c\ t|$

**by** (auto simp: abs-le-eq)

**hence** *obs*:  $|c\ \tau| < \max 1\ |c\ t| + |c\ t|$

**by** (simp add: abs-le-eq)

**have**  $|c\ \tau \cdot x_1 - c\ \tau \cdot x_2| = |c\ \tau| \cdot |x_1 - x_2|$

**by** (metis abs-mult left-diff-distrib mult.commute)

**also have**  $\dots \leq (\max 1\ |c\ t| + |c\ t|) \cdot |x_1 - x_2|$

**using** mult-abs-right-mono[OF *obs*] **by** blast

**finally have**  $|c\ \tau \cdot x_1 - c\ \tau \cdot x_2| \leq (\max 1\ |c\ t| + |c\ t|) \cdot |x_1 - x_2|$  **}}**

**hence**  $\exists L. \forall t \in cball\ t\ (\delta/2) \cap T. 0 \leq L \wedge$

$(\forall x_1 \in cball\ x\ (\delta/2). \forall x_2 \in cball\ x\ (\delta/2). |c\ t \cdot x_1 - c\ t \cdot x_2| \leq L \cdot |x_1 - x_2|)$

**by** (rule-tac  $x=\max 1\ |c\ t| + |c\ t|$  **in** exI, clarsimp simp: dist-norm)

**thus**  $\exists u > 0. \exists L. \forall t \in cball\ t\ u \cap T. 0 \leq L \wedge$

$(\forall x a \in cball\ x\ u. \forall y \in cball\ x\ u. |c\ t \cdot x a - c\ t \cdot y| \leq L \cdot |x a - y|)$

**apply**(rule-tac  $x=\delta/2$  **in** exI)

**using** d-hyp **by** auto

**qed**

**lemma** *picard-lindelof-first-order-linear*:  $t_0 \in T \implies \text{open } T \implies \text{is-interval } T \implies$

*continuous-on*  $T\ c \implies \text{picard-lindelof } (\lambda t\ x::real. c\ t * x)\ T\ UNIV\ t_0$

**apply**(unfold-locales; clarsimp?)

**apply**(intro continuous-intros, assumption)

**by** (rule local-lipschitz-first-order-linear)

**lemma**  $[\lambda s::real^2. s\$1 + s\$2 = 0] \leq$

*wp*  $(x' = (\lambda t\ s. (\chi\ i. \text{if } i=1 \text{ then } A*(s\$1)^2 + B*(s\$1) \text{ else } A*(s\$2)*(s\$1) + B*(s\$2)))) \ \&\ G \text{ on } (\lambda s. UNIV) UNIV\ @\ 0)$

$[\lambda s. 0 = -\ s\$1 - s\$2]$

**proof**—

**have** *key*: *diff-invariant*  $(\lambda s. s\ \$\ 1 + s\ \$\ 2 = 0)$

$(\lambda t\ s. \chi\ i. \text{if } i = 1 \text{ then } A*(s\$1)^2 + B*(s\$1) \text{ else } A*(s\$2)*(s\$1) + B*(s\$2))\ (\lambda s. UNIV)\ UNIV\ 0\ G$

**proof**(*clarsimp simp: diff-invariant-eq ivp-sols-def forall-2*)  
**fix**  $X::\text{real} \Rightarrow \text{real}^2$  **and**  $t::\text{real}$   
**let**  $?c = (\lambda t. X\ t\ \$\ 1 + X\ t\ \$\ 2)$   
**assume** *init: ?c 0 = 0*  
**and**  $D1: D\ (\lambda t. X\ t\ \$\ 1) = (\lambda t. A \cdot (X\ t\ \$\ 1)^2 + B \cdot X\ t\ \$\ 1)$  *on UNIV*  
**and**  $D2: D\ (\lambda t. X\ t\ \$\ 2) = (\lambda t. A \cdot X\ t\ \$\ 2 \cdot X\ t\ \$\ 1 + B \cdot X\ t\ \$\ 2)$  *on UNIV*  
**hence**  $D\ ?c = (\lambda t. ?c\ t\ * (A \cdot (X\ t\ \$\ 1) + B))$  *on UNIV*  
**by** (*auto intro!: poly-derivatives simp: field-simps power2-eq-square*)  
**hence**  $D\ ?c = (\lambda t. (A \cdot X\ t\ \$\ 1 + B) \cdot (X\ t\ \$\ 1 + X\ t\ \$\ 2))$  *on*  $\{0 \dashv\dashv t\}$   
**using** *has-vderiv-on-subset[OF - subset-UNIV[of {0 \dashv\dashv t}]]* **by** (*simp add: mult.commute*)  
**moreover** **have** *continuous-on UNIV*  $(\lambda t. A \cdot (X\ t\ \$\ 1) + B)$   
**apply**(*rule vderiv-on-continuous-on*)  
**using**  $D1$  **by** (*auto intro!: poly-derivatives simp: field-simps power2-eq-square*)  
**moreover** **have**  $D\ (\lambda t. 0) = (\lambda t. (A \cdot X\ t\ \$\ 1 + B) \cdot 0)$  *on*  $\{0 \dashv\dashv t\}$   
**by** (*auto intro!: poly-derivatives*)  
**moreover** **note** *picard-lindelof.ivp-unique-solution[OF*  
*picard-lindelof-first-order-linear[OF UNIV-I open-UNIV is-interval-univ calculation(2)]*  
*UNIV-I is-interval-closed-segment-1 subset-UNIV -*  
*ivp-solsI[OF - funcset-UNIV, of ?c]*  
*ivp-solsI[OF - funcset-UNIV, of  $\lambda t. 0$ ], of  $t\ \lambda s. 0\ 0\ \lambda s. t\ 0$ ]*  
**ultimately** **show**  $X\ t\ \$\ 1 + X\ t\ \$\ 2 = 0$   
**using** *init* **by** *auto*  
**qed**  
**show** *?thesis*  
**apply**(*subgoal-tac*  $(\lambda s. 0 = -\ s\$1 - s\$2) = (\lambda s. s\$1 + s\$2 = 0)$ , *erule ssubst*)  
**using** *key* **by** *auto*  
**qed**

## Dynamics: Fractional Darboux equality

**lemma** *fractional-darboux-arith:*

**assumes**  $x2 \neq 0$   
**shows**  $(A \cdot x1^2 + B \cdot (x1::\text{real})) / x2^2 + (A \cdot x1 + B) / x2$   
 $= (x1 \cdot (A \cdot x1 + B) + x2 \cdot (A \cdot x1 + B)) / x2^2$  (**is** *?lhs = ?rhs*)  
**proof**–  
**have** *?lhs*  $= (A \cdot x1^2 + B \cdot x1) / x2^2 + (A \cdot x1 + B) \cdot x2 / x2^2$   
**by** (*simp add: mult.commute power2-eq-square*)  
**also** **have**  $\dots = (A \cdot x1^2 + B \cdot x1 + (A \cdot x1 + B) \cdot x2) / x2^2$   
**using** *assms* **by** (*simp add: field-simps*)  
**also** **have**  $\dots = ?rhs$   
**by** (*simp add: field-simps power2-eq-square*)  
**finally** **show** *?thesis* .  
**qed**

**lemma**  $[\lambda s::\text{real}^2. s\$1 + s\$2 = 0] \leq$

*wp*  $(x' = (\lambda s. (\chi\ i. \text{if } i=1 \text{ then } (A*(s\$1)^2+B*(s\$1))/(s\$2)^2 \text{ else } (A*(s\$1)+B)/s\$2)) \ \& \ (\lambda s. y = (s\$1)^2 \wedge (s\$2)^2 > 0))$   
 $[\lambda s. s\$1 + s\$2 = 0]$

**proof**–

**have** *diff-invariant*  $(\lambda s::\text{real}^2. s\$1 + s\$2 = 0)$   
 $(\lambda t\ s. (\chi\ i. \text{if } i=1 \text{ then } (A*(s\$1)^2+B*(s\$1))/(s\$2)^2 \text{ else } (A*(s\$1)+B)/s\$2))\ (\lambda s. \{t. t \geq 0\})$   
 $UNIV\ 0\ (\lambda s. y = (s\$1)^2 \wedge (s\$2)^2 > 0)$   
**apply**(*clarsimp simp: diff-invariant-eq forall-2 ivp-sols-def*)  
**subgoal** **for**  $X\ t$   
**apply**(*rule unique-on-bounded-closed.unique-solution[of 0 {0 \dashv\dashv t} -*  
 $\lambda t\ \tau. (\tau * (A * X\ t\ \$\ 1 + B))/(X\ t\ \$\ 2)^2$   
 $UNIV - \lambda t. X\ t\ \$\ 1 + X\ t\ \$\ 2\ \lambda t. 0]$ , *simp-all add: solves-ode-def*)  
**defer**  
**apply**(*rule poly-derivatives*)

```

    apply(rule has-vderiv-on-subset, assumption, simp add: closed-segment-eq-real-ivl subset-eq)
    apply(rule has-vderiv-on-subset, assumption, simp add: closed-segment-eq-real-ivl subset-eq)
    apply(subst fractional-darboux-arith, simp-all add: closed-segment-eq-real-ivl)
    apply(rule poly-derivatives)
    apply(unfold-locales, simp-all)
  sorry
done
thus ?thesis
  by simp
qed

```

```

lemma [ $\lambda s::\text{real}^2. s\$1 + s\$2 = 0$ ]  $\leq$ 
  wp ( $x' = (\lambda s. (\chi i. \text{if } i=1 \text{ then } (A*(s\$1)^2 + B*(s\$1))/(s\$2)^2 \text{ else } (A*(s\$1)+B)/s\$2)) \ \& \ (\lambda s. y = (s\$1)^2 \wedge (s\$2)^2 > 0)$ )
  [ $\lambda s. s\$1 + s\$2 = 0$ ]
proof–
  have diff-invariant ( $\lambda s::\text{real}^2. s\$1 + s\$2 = 0$ )
    ( $\lambda t s. (\chi i. \text{if } i=1 \text{ then } (A*(s\$1)^2 + B*(s\$1))/(s\$2)^2 \text{ else } (A*(s\$1)+B)/s\$2)) \ (\lambda s. \{t. t \geq 0\})$ 
    UNIV 0 ( $\lambda s. y = (s\$1)^2 \wedge (s\$2)^2 > 0$ )
  apply(clarsimp simp: diff-invariant-eq forall-2 ivp-sols-def)
  subgoal for  $X \ t$ 
    apply(rule picard-lindelof.ivp-unique-solution[ $\text{of } \lambda t \tau. (\tau * (A * X \ t \ \$ \ 1 + B))/(X \ t \ \$ \ 2)^2 \ \{0 \dashv\dashv t\}$ 
    UNIV 0 0  $\lambda s. \{0 \dashv\dashv t\}$ ,
      where  $Y_1 = \lambda t. X \ t \ \$ \ 1 + X \ t \ \$ \ 2$  and  $Y_2 = \lambda t. 0$ ], simp-all add: ivp-sols-def)
  subgoal
    apply(unfold-locales, simp-all)
    prefer 2
    apply(clarify, rule vderiv-on-continuous-on)
    apply(auto intro!: poly-derivatives simp: closed-segment-eq-real-ivl)[1]
    apply(rule has-vderiv-on-subset, force, force)
    apply(rule has-vderiv-on-subset, force, force)
    subgoal sorry
    subgoal sorry
    done
    apply(auto intro!: poly-derivatives simp: closed-segment-eq-real-ivl)
    apply(rule has-vderiv-on-subset, force, force)
    apply(rule has-vderiv-on-subset, force, force)
    by (subst fractional-darboux-arith, simp-all add: closed-segment-eq-real-ivl)
  done
thus ?thesis
  by simp
qed

```

### Dynamics: Darboux inequality

**abbreviation** *darboux-ineq-f* ::  $\text{real}^2 \Rightarrow \text{real}^2 \ (f)$   
**where**  $f \ s \equiv (\chi i. \text{if } i=1 \text{ then } (s\$1)^2 \text{ else } (s\$2)*(s\$1)+(s\$1)^2)$

**abbreviation** *darboux-ineq-flow2* ::  $\text{real} \Rightarrow \text{real}^2 \Rightarrow \text{real}^2 \ (\varphi)$   
**where**  $\varphi \ t \ s \equiv (\chi i. \text{if } i=1 \text{ then } (s\$1/(1 - t * s\$1)) \text{ else } (s\$2 - s\$1 * \ln(1 - t * s\$1))/(1 - t * s\$1))$

**lemma** *darboux-flow-ivp*:  $(\lambda t. \varphi \ t \ s) \in \text{Sols } (\lambda t. f) \ (\lambda s. \{t. 0 \leq t \wedge t * s \ \$ \ 1 < 1\}) \ \text{UNIV } 0 \ s$   
**by** (rule ivp-solsI) (auto intro!: poly-derivatives  
 simp: forall-2 power2-eq-square add-divide-distrib power-divide vec-eq-iff)

**lemma** *darboux-picard*: *picard-lindelof*  $(\lambda t. f) \ \text{UNIV } \text{UNIV } 0$   
**apply** unfold-locales  
**apply** (simp-all add: local-lipschitz-def lipschitz-on-def)

**apply** (*clarsimp simp: dist-norm norm-vec-def L2-set-def*)  
**unfolding** *UNIV-2*  
**sorry**

**lemma** *darboux-ineq-arith:*

**assumes**  $0 \leq s_1 + s_2$  **and**  $0 \leq (t::real)$  **and**  $t * s_1 < 1$   
**shows**  $0 \leq s_1 / (1 - t * s_1) + (s_2 - s_1 * \ln (1 - t * s_1)) / (1 - t * s_1)$

**proof**—

**have**  $s_1 * \ln (1 - t * s_1) \leq 0$

**proof**(*cases*  $s_1 \leq 0$ )

**case** *True*

**hence**  $1 - t * s_1 \geq 1$

**using**  $\langle 0 \leq t \rangle$  **by** (*simp add: mult-le-0-iff*)

**thus** *?thesis*

**using** *True ln-ge-zero mult-nonneg-nonpos2* **by** *blast*

**next**

**case** *False*

**hence**  $1 - t * s_1 \leq 1$

**using**  $\langle 0 \leq t \rangle$  **by** *auto*

**thus** *?thesis*

**by** (*metis (mono-tags, hide-lams) False add.left-neutral antisym-conv assms(3)*  
*diff-le-eq ln-ge-zero-iff ln-one mult-zero-right not-le order-refl zero-le-mult-iff*)

**qed**

**hence**  $s_1 + s_2 - s_1 * \ln (1 - t * s_1) \geq s_1 + s_2$

**by** *linarith*

**hence**  $(s_1 + s_2 - s_1 * \ln (1 - t * s_1)) / (1 - t * s_1) \geq (s_1 + s_2) / (1 - t * s_1)$

**using**  $\langle t * s_1 < 1 \rangle$  **by** (*simp add:  $\langle 0 \leq s_1 + s_2 \rangle$  frac-le*)

**also have**  $(s_1 + s_2) / (1 - t * s_1) \geq 0$

**using**  $\langle t * s_1 < 1 \rangle$  **by** (*simp add:  $\langle 0 \leq s_1 + s_2 \rangle$* )

**ultimately show** *?thesis*

**by** (*metis (full-types) add-diff-eq add-divide-distrib order-trans*)

**qed**

**lemma**  $\lceil \lambda s::real^2. s\$1 + s\$2 \geq 0 \rceil \leq$

*wp (x' = ( $\lambda t. f$ ) & ( $\lambda s. y = (s\$1)^2$ ) on ( $\lambda s. \{t. 0 \leq t \wedge t * s \$ 1 < 1\}$ ) UNIV @ 0)*

$\lceil \lambda s. s\$1 + s\$2 \geq 0 \rceil$

**unfolding** *g-ode-def*

**apply**(*subst picard-lindelof.g-orbital-orbit[OF darboux-picard - - darboux-flow-ivp]*)

**unfolding** *g-evol-def[symmetric]* *wp-g-dyn* **apply**(*simp-all add: is-interval-def*)

**apply** (*smt mult-less-cancel-right*)

**using** *darboux-ineq-arith* **by** *smt*

**lemma**  $\lceil \lambda s::real^2. s\$1 + s\$2 \geq 0 \rceil \leq$

*wp (EVOL  $\varphi$  ( $\lambda s. y = (s\$1)^2$ ) ( $\lambda s. \{t. 0 \leq t \wedge t * s \$ 1 < 1\}$ ))*

$\lceil \lambda s. s\$1 + s\$2 \geq 0 \rceil$

**apply**(*subst wp-g-dyn, simp-all*)

**using** *darboux-ineq-arith* **by** *smt*

**no-notation** *darboux-ineq-flow2* ( $\varphi$ )

**and** *darboux-ineq-f* ( $f$ )

## Dynamics: Bifurcation

**lemma** *picard-lindelof-dyn-bif:*

*continuous-on T (g::real  $\Rightarrow$  real)  $\Rightarrow$   $t_0 \in T \Rightarrow$  is-interval T  $\Rightarrow$  open T  $\Rightarrow$  picard-lindelof ( $\lambda t \tau::real. r + \tau^2$ ) T UNIV  $t_0$*

**proof**(*unfold-locales; clarsimp simp: dist-norm local-lipschitz-def lipschitz-on-def*)

**fix**  $x t::real$



```

{fix x1 and x2
  assume x1 ∈ cball x 1 and x2 ∈ cball x 1
  hence leq: |x - x1| ≤ 1 |x - x2| ≤ 1
  by (auto simp: dist-norm)
  have |x1 + x2| = |x1 - x + x2 - x + 2 * x|
  by simp
  also have ... ≤ |x1 - x| + |x2 - x| + 2 * |x|
  using abs-triangle-ineq by auto
  also have ... ≤ 2 * (1 + |x|)
  using leq by auto
  finally have obs: |x1 + x2| ≤ 2 * (1 + |x|) .
  also have |x12 - x22| = |x1 + x2| * |x1 - x2|
  by (metis abs-mult power2-eq-square square-diff-square-factored)
  ultimately have |x12 - x22| ≤ (2 * (1 + |x|)) * |x1 - x2|
  by (metis abs-ge-zero mult-right-mono)}
thus ∃ u > 0. (∃ τ. |t - τ| ≤ u ∧ τ ∈ T) ⟶ (∃ L ≥ 0. ∀ xa ∈ cball x u. ∀ y ∈ cball x u. |xa2 - y2| ≤ L · |xa - y|)
by (rule-tac x=1 in exI, clarsimp, rule-tac x=2 · (1 + |x|) in exI, auto)
qed

```

```

lemma r ≤ 0 ⟹ ∃ c. [λs::real^1. s$1 = c] ≤
  wp (x' = (λt s. (χ i. r + (s$1)^2)) & G on (λs. UNIV) UNIV @ 0)
  [λs. s$1 = c]
proof (rule-tac x=sqrt |r| in exI, clarsimp simp: diff-invariant-eq ivp-sols-def)
  fix X::real⇒real^1 and t::real
  assume init: X 0 $ 1 = sqrt (- r) and r ≤ 0
  and D1: D (λx. X x $ 1) = (λx. r + (X x $ 1)2) on UNIV
  hence D (λx. X x $ 1) = (λx. r + (X x $ 1)2) on {0--t}
  using has-vderiv-on-subset by blast
  moreover have continuous-on UNIV (λt. X t $ 1)
  apply (rule vderiv-on-continuous-on)
  using D1 by assumption
  moreover have key: D (λt. sqrt (- r)) = (λt. r + (sqrt (- r))2) on {0--t}
  apply (subgoal-tac (λt. r + (sqrt (- r))2) = (λt. 0))
  apply (erule ssubst, rule poly-derivatives)
  using ⟨r ≤ 0⟩ by auto
  moreover note picard-lindelof.ivp-unique-solution[OF
    picard-lindelof-dyn-bif[OF calculation(2) UNIV-I is-interval-univ open-UNIV]
    UNIV-I is-interval-closed-segment-1 subset-UNIV -
    ivp-solsI[of λx. X x $ 1 - λs. {0--t} sqrt (- r) 0, OF - - funcset-UNIV]
    ivp-solsI[of λt. sqrt (- r) -, OF - - funcset-UNIV], of t r]
  ultimately show X t $ 1 = sqrt (- r)
  using ⟨r ≤ 0⟩ init by auto
qed

```

### Dynamics: Parametric switching between two different damped oscillators

```

lemma exhaust-5:
  fixes x :: 5
  shows x = 1 ∨ x = 2 ∨ x = 3 ∨ x = 4 ∨ x = 5
proof (induct x)
  case (of-int z)
  then have 0 ≤ z and z < 5 by simp-all
  then have z = 0 ∨ z = 1 ∨ z = 2 ∨ z = 3 ∨ z = 4 by arith
  then show ?case by auto
qed

```

```

lemma forall-5: (∀ i::5. P i) = (P 1 ∧ P 2 ∧ P 3 ∧ P 4 ∧ P 5)
  by (metis exhaust-5)

```

**abbreviation** *switch-two-osc-f* ::  $\text{real}^5 \Rightarrow \text{real}^5$  (*f*)

**where**  $f\ s \equiv (\chi\ i.\ \text{if } i = 4 \text{ then } s\$5 \text{ else } (\text{if } i = 5 \text{ then } -(s\$3^2) * s\$4 - 2 * s\$2 * s\$3 * s\$5 \text{ else } 0))$

**declare** *wp-diff-inv* [*simp del*]

**lemma**  $-2 \leq a \implies a \leq 2 \implies b^2 \geq 1/3 \implies$

$[\lambda s. s\$3 \geq 0 \wedge s\$2 \geq 0 \wedge s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1] \leq wp$

(*LOOP*

$((x' = (\lambda t. f) \ \& \ (\lambda s. \text{True})) \text{ on } (\lambda s. \{0..\}) \text{ UNIV } @ 0);$

$(([\lambda s. s\$4 = s\$5 * a]; (3 ::= (\lambda s. 2 * s\$3)); (2 ::= (\lambda s. s\$2/2)); (1 ::= (\lambda s. s\$1 * ((2 \cdot s\$3)^2 + 1)/(s\$3^2 + 1)))) \cup$

$([\lambda s. s\$4 = s\$5 * b]; (3 ::= (\lambda s. s\$3/2)); (2 ::= (\lambda s. 2 * s\$2)); (1 ::= (\lambda s. s\$1 * ((s\$3)^2 + 1)/(2 * (s\$3^2 + 1)))) \cup$

$[\lambda s. \text{True}]))$

*INV*  $(\lambda s. s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0))$

$[\lambda s. s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1]$

**apply**(*subst change-loopI*[**where**  $I = \lambda s. s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$ ])

**apply**(*rule wp-loopI*, *simp-all add: le-wp-choice-iff*, *intro conjI*)

**apply**(*subst g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$ ])

**apply**(*rule wp-g-odei*, *simp*)

**apply**(*rule-tac C* =  $\lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$  **in** *diff-cut-rule*, *simp-all*)

**apply**(*subst g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$ ], *rule wp-g-odei*; *simp add: wp-diff-inv*)

**apply**(*rule diff-invariant-conj-rule*)

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**apply**(*simp add: wp-diff-inv*, *intro diff-invariant-conj-rule*)

**apply**(*rule-tac*  $\nu' = \lambda s. -4 * (s\$2) * (s\$3) * (s\$5)^2$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2); (*clarsimp simp: forall-5*)?)

**apply**(*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)[1]

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**subgoal sorry**

**apply**(*subst g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$3^2 * s\$4^2 + s\$5^2 \leq s\$1 \wedge s\$2 \geq 0 \wedge s\$3 \geq 0$ ])

**apply**(*rule wp-g-odei*, *simp*)

**apply**(*rule-tac C* =  $\lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$  **in** *diff-cut-rule*, *simp-all*)

**apply**(*subst g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$ ], *rule wp-g-odei*, *simp-all*)

**apply**(*simp add: wp-diff-inv*, *rule diff-invariant-conj-rule*)

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**apply**(*simp add: wp-diff-inv*, *intro diff-invariant-conj-rule*)

**apply**(*rule-tac*  $\nu' = \lambda s. -4 * (s\$2) * (s\$3) * (s\$5)^2$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2); (*clarsimp simp: forall-5*)?)

**apply**(*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)[1]

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**subgoal sorry**

**apply**(*rule-tac C* =  $\lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$  **in** *diff-cut-rule*, *simp-all*)

**apply**(*subst g-ode-inv-def*[*symmetric*, **where**  $I = \lambda s. s\$2 \geq 0 \wedge s\$3 \geq 0$ ], *rule wp-g-odei*, *simp-all*)

**apply**(*simp add: wp-diff-inv*, *rule diff-invariant-conj-rule*)

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**apply**(*simp add: wp-diff-inv*, *intro diff-invariant-conj-rule*)

**apply**(*rule-tac*  $\nu' = \lambda s. -4 * (s\$2) * (s\$3) * (s\$5)^2$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2); (*clarsimp simp: forall-5*)?)

**apply**(*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)[1]

**apply** (*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 0$  **in** *diff-invariant-rules*(2), *simp-all add: forall-5*) +

**done**

**declare** *wp-diff-inv* [*simp*]

**no-notation** *switch-two-osc-f* (*f*)

### Dynamics: Nonlinear 1

**lemma**  $\lceil \lambda s :: \text{real}^1. s^1^3 \geq -1 \rceil \leq wp$   
 $(x' = (\lambda s. \chi i. (s^1 - 3)^4 + a) \ \& \ (\lambda s. a \geq 0))$   
 $\lceil \lambda s. s^1^3 \geq -1 \rceil$   
**apply**(*simp*, *rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 3 * s^1^2 * ((s^1 - 3)^4 + a)$  **in** *diff-invariant-rules*(2))  
**by** (*auto intro!*: *poly-derivatives simp: field-simps*)

### Dynamics: Nonlinear 2

**lemma**  $\lceil \lambda s :: \text{real}^2. s^1 + (s^2^2)/2 = a \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s^1 * s^2 \text{ else } -s^1) \ \& \ G)$   
 $\lceil \lambda s. s^1 + (s^2^2)/2 = a \rceil$   
**by** (*auto intro!*: *diff-invariant-rules poly-derivatives*)

### Dynamics: Nonlinear 4

**lemma**  $\lceil \lambda s :: \text{real}^2. (s^1)^2/2 - (s^2^2)/2 \geq a \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s^2 + s^1 * (s^2^2) \text{ else } -s^1 + s^1^2 * s^2) \ \& \ (\lambda s. s^1 \geq 0 \wedge s^2 \geq 0))$   
 $\lceil \lambda s. (s^1)^2/2 - (s^2^2)/2 \geq a \rceil$   
**apply**(*simp*, *rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. s^1 * (s^2 + s^1 * (s^2^2)) - s^2 * (-s^1 + s^1^2 * s^2)$  **in** *diff-invariant-rules*(2))  
**by** (*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)

### Dynamics: Nonlinear 5

**lemma**  $\lceil \lambda s :: \text{real}^2. -(s^1) * (s^2) \geq a \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s^1 - s^2 + s^1 * s^2 \text{ else } -s^2 - s^2^2) \ \& \ G)$   
 $\lceil \lambda s. -(s^1) * (s^2) \geq a \rceil$   
**apply**(*simp*, *rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. (-s^1 + s^2 - s^1 * s^2) * s^2 - s^1 * (-s^2 - s^2^2)$  **in** *diff-invariant-rules*(2))  
**by** (*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)

### Dynamics: Riccati

**lemma**  $\lceil \lambda s :: \text{real}^1. 2 * s^1^3 \geq 1/4 \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. s^1^2 + s^1^4) \ \& \ G)$   
 $\lceil \lambda s. 2 * s^1^3 \geq 1/4 \rceil$   
**apply**(*simp*, *rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 24 * (s^1^2) * (s^1^2 + s^1^4)$  **in** *diff-invariant-rules*(2); *clarsimp*)  
**by** (*auto intro!*: *poly-derivatives simp: field-simps power2-eq-square*)

### Dynamics: Nonlinear differential cut

**lemma**  $\lceil \lambda s :: \text{real}^2. s^1^3 \geq -1 \wedge s^2^5 \geq 0 \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } (s^1 - 3)^4 \text{ else } s^2^2) \ \& \ G)$   
 $\lceil \lambda s. s^1^3 \geq -1 \wedge s^2^5 \geq 0 \rceil$   
**apply**(*simp*, *rule* *diff-invariant-rules*)  
**apply**(*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. 3 * s^1^2 * (s^1 - 3)^4$  **in** *diff-invariant-rules*(2))  
**apply**(*simp-all add: forall-2, force intro!*: *poly-derivatives*)  
**apply**(*rule-tac*  $\nu' = \lambda s. 0$  **and**  $\mu' = \lambda s. s^2^2$  **in** *diff-invariant-rules*(2))  
**by** (*auto intro!*: *diff-invariant-rules poly-derivatives simp: forall-2*)

### STTT Tutorial: Example 1

**lemma**  $A > 0 \implies \lceil \lambda s :: \text{real}^2. s^2 \geq 0 \rceil \leq$   
 $wp (x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s^2 \text{ else } A) \ \& \ G)$

```

[λs. s$2 ≥ 0]
apply(subst local-flow.wp-g-ode-subset[where T=UNIV
  and φ=λt s. χ i::2. if i=1 then A * t^2/2 + s$2 * t + s$1 else A * t + s$2])
  apply(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)
  apply(clarsimp, rule-tac x=1 in exI)+
apply(clarsimp simp: dist-norm norm-vec-def L2-set-def)
unfolding UNIV-2 using exhaust-2 by (auto intro!: poly-derivatives simp: vec-eq-iff)

```

## STTT Tutorial: Example 2

```

lemma local-flow-STTT-Ex2:
  local-flow (λs::real^3. χ i. if i = 1 then s$2 else (if i=2 then s$3 else 0)) UNIV UNIV
  (λt s. χ i. if i = 1 then s$3 * t^2/2 + s$2 * t + s$1 else (if i=2 then s$3 * t + s$2 else s$i))
  apply(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)
  apply(clarsimp, rule-tac x=1 in exI)+
  apply(clarsimp simp: dist-norm norm-vec-def L2-set-def)
  unfolding UNIV-3 by (auto intro!: poly-derivatives simp: forall-3 vec-eq-iff)

```

```

lemma A > 0 ⟹ B > 0 ⟹ [λs::real^3. s$2 ≥ 0] ≤ wp
  (LOOP (
    (((3 ::= (λs. A)) ∪ (3 ::= (λs. 0)) ∪ (3 ::= (λs. B)))));
    (x' = (λs. χ i. if i=1 then s$2 else (if i=2 then s$3 else 0)) & (λs. s$2 ≥ 0)))
  ) INV (λs. s$2 ≥ 0)
  [λs. s$2 ≥ 0]
  apply(rule wp-loopI, simp-all add: le-wp-choice-iff, intro conjI)
  by (simp-all add: local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])

```

## STTT Tutorial: Example 3a

```

lemma STTexample3a-arith:
  assumes 0 < (B::real) 0 ≤ t 0 ≤ x2 and key: x1 + x2^2 / (2 · B) ≤ S
  shows x2 · t - B · t^2 / 2 + x1 + (x2 - B · t)^2 / (2 · B) ≤ S (is ?lhs ≤ S)
proof–
  have ?lhs = 2 * B * x2 · t / (2*B) - B^2 · t^2 / (2*B) + (2 * B * x1) / (2*B) + (x2 - B · t)^2 / (2 · B)
    using ⟨0 < B⟩ by (auto simp: power2-eq-square)
  also have (x2 - B · t)^2 / (2 · B) = x2^2 / (2*B) + B^2 * t^2 / (2*B) - 2*x2*B*t / (2*B)
    using ⟨0 < B⟩ by (auto simp: power2-diff field-simps)
  ultimately have ?lhs = x1 + x2^2 / (2 · B)
    using ⟨0 < B⟩ by auto
  thus ?lhs ≤ S
    using key by simp
qed

```

```

lemma A > 0 ⟹ B > 0 ⟹ [λs::real^3. s$2 ≥ 0 ∧ s$1 + s$2^2/(2*B) < S] ≤ wp
  (LOOP (
    (([λs. s$1 + s$2^2/(2*B) < S]; (3 ::= (λs. A))) ∪ ([λs. s$2 = 0]; (3 ::= (λs. 0)))) ∪ (3 ::= (λs. -
    B)))));
    ((x' = (λs. χ i. if i=1 then s$2 else (if i=2 then s$3 else 0)) & (λs. s$2 ≥ 0 ∧ s$1 + s$2^2/(2*B) ≤
    S)) ∪
    (x' = (λs. χ i. if i=1 then s$2 else (if i=2 then s$3 else 0)) & (λs. s$2 ≥ 0 ∧ s$1 + s$2^2/(2*B) ≥
    S)))
  ) INV (λs. s$2 ≥ 0 ∧ s$1 + s$2^2/(2*B) ≤ S)
  [λs. s$1 ≤ S]
  apply(rule wp-loopI)
  apply(simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])
  apply safe
  apply (smt not-sum-power2-lt-zero zero-compare-simps(5))
  apply(erule-tac x=0 in allE)
  by (auto simp: STTexample3a-arith)

```

**STTT Tutorial: Example 4a**

**lemma**  $A > 0 \implies [\lambda s::\text{real}^3. s\$2 \leq V] \leq wp$   
 (LOOP  
 ( $[\lambda s. s\$2 = V]; (\mathcal{I} ::= (\lambda s. 0))$ )  $\cup$  ( $[\lambda s. s\$2 \neq V]; (\mathcal{I} ::= (\lambda s. A))$ );  
 ( $x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \leq V)$ )  
 INV ( $\lambda s. s\$2 \leq V$ )  
 $[\lambda s. s\$2 \leq V]$   
 by (rule wp-loopI)  
 (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])

**STTT Tutorial: Example 4b**

**lemma**  $A > 0 \implies [\lambda s::\text{real}^3. s\$2 \leq V] \leq wp$   
 (LOOP  
 ( $\mathcal{I} ::= (\lambda s. A)$ );  
 ( $x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \leq V)$ )  
 INV ( $\lambda s. s\$2 \leq V$ )  
 $[\lambda s. s\$2 \leq V]$   
 by (rule wp-loopI) (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])

**STTT Tutorial: Example 4c**

**lemma**  $A > 0 \implies [\lambda s::\text{real}^3. s\$2 \leq V] \leq wp$   
 (LOOP  
 ( $[\lambda s. s\$2 = V]; (\mathcal{I} ::= (\lambda s. 0))$ )  $\cup$  ( $[\lambda s. s\$2 \neq V]; (\mathcal{I} ::= (\lambda s. A))$ );  
 ( $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \leq V)) \cup$   
 ( $x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \geq V)$ ))  
 INV ( $\lambda s. s\$2 \leq V$ )  
 $[\lambda s. s\$2 \leq V]$   
 apply (rule wp-loopI)  
 apply (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])  
 by (clarsimp, erule-tac  $x=0$  in allE, auto)

**STTT Tutorial: Example 5**

**lemma** STTexample5-arith:  
 assumes  $0 < A$   $0 < B$   $0 < \varepsilon$   $0 \leq x2$   $0 \leq (t::\text{real})$   
 and key:  $x1 + x2^2 / (2 \cdot B) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2) / B + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2)) \leq S$  (is ?k3  $\leq S$ )  
 and ghyp:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon$   
 shows  $A \cdot t^2 / 2 + x2 \cdot t + x1 + (A \cdot t + x2)^2 / (2 \cdot B) \leq S$  (is ?k0  $\leq S$ )  
 proof—  
 have  $t \leq \varepsilon$   
 using ghyp  $\langle 0 \leq t \rangle$  by auto  
 hence  $A \cdot t^2 / 2 + t \cdot x2 \leq A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2$   
 using  $\langle 0 \leq t \rangle \langle 0 < A \rangle \langle 0 \leq x2 \rangle$   
 by (smt field-sum-of-halves mult-right-mono power-less-imp-less-base real-mult-le-cancel-iff2)  
 hence  $((A + B)/B) * (A \cdot t^2 / 2 + t \cdot x2) + x1 + x2^2 / (2 \cdot B) \leq$   
 $x1 + x2^2 / (2 \cdot B) + ((A + B)/B) * (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2)$  (is ?k1  $\leq$  ?k2)  
 using  $\langle 0 < B \rangle \langle 0 < A \rangle$  by (smt real-mult-le-cancel-iff2 zero-compare-simps(9))  
 moreover have ?k0 = ?k1  
 using  $\langle 0 < B \rangle \langle 0 < A \rangle$  by (auto simp: field-simps power2-sum power2-eq-square)  
 moreover have ?k2 = ?k3  
 using  $\langle 0 < B \rangle \langle 0 < A \rangle$  by (auto simp: field-simps power2-sum power2-eq-square)  
 ultimately show ?k0  $\leq S$   
 using key by linarith  
 qed

**lemma** local-flow-STTT-Ex5:

local-flow ( $\lambda s::\text{real}^4. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1))$ ) UNIV UNIV

( $\lambda t s. \chi i. \text{if } i = 1 \text{ then } s\$3 * t^2/2 + s\$2 * t + s\$1 \text{ else (if } i=2 \text{ then } s\$3 * t + s\$2 \text{ else (if } i=3 \text{ then } s\$3 \text{ else } t+s\$4))$ )  
**apply**(*unfold-locales*, *simp-all add: local-lipschitz-def forall-2 lipschitz-on-def*)  
**apply**(*clarsimp*, *rule-tac x=1 in exI*) +  
**apply**(*clarsimp simp: dist-norm norm-vec-def L2-set-def*)  
**unfolding** *UNIV-4* **by** (*auto intro!: poly-derivatives simp: forall-4 vec-eq-iff*)

**lemma**  $A > 0 \implies B > 0 \implies \varepsilon > 0 \implies [\lambda s::\text{real}^4. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*B) \leq S] \leq wp$   
*(LOOP*  
 (  
 ( $[\lambda s. s\$1 + s\$2^2/(2*B) + (A/B + 1) * (A/2 * \varepsilon^2 + \varepsilon * s\$2) \leq S]; (\mathcal{I} ::= (\lambda s. A)) \cup$   
 $[\lambda s. s\$2 = 0]; (\mathcal{I} ::= (\lambda s. 0))) \cup$   
 $(\mathcal{I} ::= (\lambda s. -B))$ );  
 $(\mathcal{I} ::= (\lambda s. 0))$ ;  
 $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1))) \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$   
*INV* ( $\lambda s. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*B) \leq S$ )  
 $[\lambda s. s\$1 \leq S]$   
**apply** (*rule wp-loopI*)  
**apply** (*simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex5]*)  
**apply** *safe*  
**apply** (*smt not-sum-power2-lt-zero zero-compare-simps(5)*)  
**by** (*auto simp: STTexample3a-arith STTexample5-arith*)

## STTT Tutorial: Example 6

**lemma** *STTexample6-arith*:

**assumes**  $0 < A \ 0 < B \ 0 < \varepsilon \ 0 \leq x2 \ 0 \leq (t::\text{real}) - B \leq k \ k \leq A$   
**and** *key*:  $x1 + x2^2 / (2 \cdot B) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2) / B + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot x2)) \leq S$  (**is** ?*k3*  $\leq S$ )  
**and** *ghyp*:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq k \cdot \tau + x2 \wedge \tau \leq \varepsilon$   
**shows**  $k \cdot t^2 / 2 + x2 \cdot t + x1 + (k \cdot t + x2)^2 / (2 \cdot B) \leq S$  (**is** ?*k0*  $\leq S$ )

**proof**–

**have**  $0 \leq k \cdot t + x2 + x2$   
**using** *ghyp*  $\langle 0 \leq x2 \rangle \langle 0 \leq t \rangle$  **by** *force*  
**hence**  $0 \leq (k \cdot t + 2 \cdot x2) \cdot t/2$   
**by** (*metis assms(5) divide-nonneg-pos is-num-normalize(1) mult-2 mult-sign-intros(1) rel-simps(51)*)  
**hence** *f1*:  $0 \leq k \cdot t^2/2 + t \cdot x2$   
**by** (*auto simp: field-simps power2-eq-square*)  
**have** *f2*:  $0 \leq (k + B) / B \ (k + B) / B \leq (A + B) / B$   
**using**  $\langle 0 < A \rangle \langle 0 < B \rangle \langle -B \leq k \rangle \langle k \leq A \rangle$  *divide-le-cancel* **by** (*auto, fastforce*)  
**have**  $t \leq \varepsilon$   
**using** *ghyp*  $\langle 0 \leq t \rangle$  **by** *auto*  
**hence**  $k \cdot t^2/2 + t \cdot x2 \leq A \cdot \varepsilon^2/2 + \varepsilon \cdot x2$   
**using**  $\langle k \leq A \rangle$  **by** (*auto simp: mult-right-mono*)  
**also** **have** *f3*:  $\dots \leq A \cdot \varepsilon^2/2 + \varepsilon \cdot x2$   
**using**  $\langle 0 \leq t \rangle \langle 0 < A \rangle \langle 0 \leq x2 \rangle \langle t \leq \varepsilon \rangle$   
**by** (*smt field-sum-of-halves mult-right-mono power-less-imp-less-base real-mult-le-cancel-iff2*)  
**finally** **have**  $k \cdot t^2/2 + t \cdot x2 \leq A \cdot \varepsilon^2/2 + \varepsilon \cdot x2$  .  
**hence**  $((k + B)/B) \cdot (k \cdot t^2/2 + t \cdot x2) \leq ((A + B)/B) \cdot (A \cdot \varepsilon^2/2 + \varepsilon \cdot x2)$   
**using** *f1 f2*  $\langle k \leq A \rangle$  **apply**(*rule-tac b=((A + B)/B) \* (A \* ε^2/2 + ε \* x2) in order.trans*)  
**apply** (*rule mult-mono', simp, simp, simp add: mult-right-mono, simp, simp*)  
**by** (*metis f3 add-sign-intros(4) assms(1,2) less-eq-real-def mult-zero-left real-mult-le-cancel-iff2 zero-compare-simps(5)*)  
**hence**  $((k + B)/B) \cdot (k \cdot t^2/2 + t \cdot x2) + x1 + x2^2 / (2 \cdot B) \leq$   
 $x1 + x2^2 / (2 \cdot B) + ((A + B)/B) \cdot (A \cdot \varepsilon^2/2 + \varepsilon \cdot x2)$  (**is** ?*k1*  $\leq$  ?*k2*)  
**using**  $\langle 0 < B \rangle \langle 0 < A \rangle$  **by** (*smt real-mult-le-cancel-iff2 zero-compare-simps(9)*)  
**moreover** **have** ?*k0* = ?*k1*  
**using**  $\langle 0 < B \rangle \langle 0 < A \rangle$  **by** (*auto simp: field-simps power2-sum power2-eq-square*)

moreover have  $?k2 = ?k3$   
 using  $\langle 0 < B \rangle \langle 0 < A \rangle$  by (auto simp: field-simps power2-sum power2-eq-square)  
 ultimately show  $?k0 \leq S$   
 using key by linarith  
 qed

**lemma**  $A > 0 \implies B > 0 \implies \varepsilon > 0 \implies [\lambda s::\text{real}^4. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*B) \leq S] \leq wp$   
 (LOOP  
 (  
 ([ $\lambda s. s\$1 + s\$2^2/(2*B) + (A/B + 1) * (A/2 * \varepsilon^2 + \varepsilon * s\$2) \leq S$ ];( $\beta ::= ?$ );[ $\lambda s. -B \leq s\$3 \wedge s\$3 \leq A$ ])  $\cup$   
 ([ $\lambda s. s\$2 = 0$ ];( $\beta ::= (\lambda s. 0)$ )))  $\cup$   
 ( $\beta ::= (\lambda s. -B)$ ));  
 ( $\beta ::= (\lambda s. 0)$ );  
 ( $x' = (\lambda s. \chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1))} \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$ )  
 INV ( $\lambda s. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*B) \leq S$ )  
 [ $\lambda s. s\$1 \leq S$ ]  
 apply (rule wp-loopI)  
 apply (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex5])  
 apply safe  
 apply (smt not-sum-power2-lt-zero zero-compare-simps(5))  
 by (auto simp: STTexample3a-arith STTexample6-arith)

### STTT Tutorial: Example 7

**lemma** STTexample7-arith1:

assumes  $(0::\text{real}) < A \ 0 < b \ 0 < \varepsilon \ 0 \leq v \ 0 \leq t \ k \leq A$   
 and  $x + v^2 / (2 \cdot b) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) / b + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v)) \leq S$  (is ?expr1  $\leq S$ )  
 and guard:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq k \cdot \tau + v \wedge \tau \leq \varepsilon$   
 shows  $k \cdot t^2 / 2 + v \cdot t + x + (k \cdot t + v)^2 / (2 \cdot b) \leq S$  (is ?lhs  $\leq S$ )

**proof**–

have obs1:  $?lhs \cdot (2 \cdot b) = k \cdot t^2 \cdot b + 2 \cdot v \cdot t \cdot b + 2 \cdot x \cdot b + (k \cdot t + v)^2$  (is - = ?expr2  $k \ t$ )  
 using  $\langle 0 < b \rangle$  by (simp add: field-simps)  
 have ?expr2  $A \ \varepsilon = ?expr1 \cdot (2 \cdot b)$   
 using  $\langle 0 < b \rangle$  by (simp add: field-simps power2-eq-square)  
 also have  $\dots \leq S \cdot (2 \cdot b)$   
 using (?expr1  $\leq S$ )  $\langle 0 < b \rangle$  by (smt real-mult-less-iff1)  
 finally have obs2:  $?expr2 \ A \ \varepsilon \leq S \cdot (2 \cdot b)$ .  
 have  $t \leq \varepsilon$   
 using guard  $\langle 0 \leq t \rangle$  by auto  
 hence  $t^2 \leq \varepsilon^2 \ k \cdot t + v \leq A \cdot \varepsilon + v$   
 using  $\langle k \leq A \rangle \langle 0 < A \rangle$  power-mono[OF  $\langle t \leq \varepsilon \rangle \langle 0 \leq t \rangle$ , of 2]  
 by auto (meson  $\langle 0 \leq t \rangle$  less-eq-real-def mult-mono)  
 hence  $k \cdot t^2 \cdot b \leq A \cdot \varepsilon^2 \cdot b \ 2 \cdot v \cdot t \cdot b \leq 2 \cdot v \cdot \varepsilon \cdot b$   
 using  $\langle t \leq \varepsilon \rangle \langle 0 < b \rangle \langle k \leq A \rangle \langle 0 \leq v \rangle$   
 by (auto simp: mult-left-mono) (meson  $\langle 0 < A \rangle$  less-eq-real-def mult-mono zero-compare-simps(12))  
 hence ?expr2  $k \ t \leq ?expr2 \ A \ \varepsilon$   
 by (smt  $\langle k \cdot t + v \leq A \cdot \varepsilon + v \rangle$  ends-in-segment(2)  $\langle 0 \leq t \rangle$  guard power-mono)  
 hence  $?lhs \cdot (2 \cdot b) \leq S \cdot (2 \cdot b)$   
 using obs1 obs2 by simp  
 thus  $?lhs \leq S$   
 using  $\langle 0 < b \rangle$  by (smt real-mult-less-iff1)

qed

**lemma** STTexample7-arith2:

assumes  $(0::\text{real}) < b \ 0 \leq v \ 0 \leq t \ k \leq -b$   
 and key:  $x + v^2 / (2 \cdot b) \leq S$   
 and guard:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq k \cdot \tau + v \wedge \tau \leq \varepsilon$

shows  $k \cdot t^2 / 2 + v \cdot t + x + (k \cdot t + v)^2 / (2 \cdot b) \leq S$  (is ?lhs  $\leq S$ )

proof—

have obs:  $1 + k/b \leq 0$   $k \cdot t + v \geq 0$

using  $\langle k \leq -b \rangle \langle 0 < b \rangle$  guard  $\langle 0 \leq t \rangle$  by (auto simp: mult-imp-div-pos-le real-add-le-0-iff)

have ?lhs =  $(k \cdot t + v + v) \cdot t / 2 \cdot (1 + k/b) + x + v^2 / (2 \cdot b)$

using  $\langle 0 < b \rangle$  by (auto simp: field-simps power2-eq-square)

also have  $\dots \leq x + v^2 / (2 \cdot b)$

using obs  $\langle 0 \leq t \rangle \langle 0 \leq v \rangle$

by (smt mult-nonneg-nonneg zero-compare-simps(11) zero-compare-simps(6))

also have  $\dots \leq S$

using key .

finally show ?thesis .

qed

lemma  $A > 0 \implies B \geq b \implies b > 0 \implies \varepsilon > 0 \implies [\lambda s::\text{real}^4. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*b) \leq S] \leq$

wp

(LOOP

(

( $[\lambda s. s\$1 + s\$2^2/(2*b) + (A/b + 1) * (A/2 * \varepsilon^2 + \varepsilon * s\$2) \leq S]; (\mathcal{B} ::= ?); [\lambda s. -B \leq s\$3 \wedge$

$s\$3 \leq A] \cup$

( $[\lambda s. s\$2 = 0]; (\mathcal{B} ::= (\lambda s. 0))) \cup$

( $(\mathcal{B} ::= ?); [\lambda s. -B \leq s\$3 \wedge s\$3 \leq -b]$ )

); ( $\mathcal{A} ::= (\lambda s. 0)$ );

( $x' = (\lambda s. \chi \text{ i. if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1))) \& (\lambda s. s\$2 \geq 0 \wedge s\$4$

$\leq \varepsilon)$ )

INV  $(\lambda s. s\$2 \geq 0 \wedge s\$1 + s\$2^2/(2*b) \leq S)$ )

$[\lambda s. s\$1 \leq S]$

apply (rule wp-loopI)

apply (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex5])

apply (safe)

apply (smt not-sum-power2-lt-zero zero-compare-simps(5))

apply (force simp: STTExample7-arith1, force)

using STTExample7-arith2[of b - \$2 - - \$1 S] by blast

### STTT Tutorial: Example 9a

lemma STTExample9a-arith:

$(10 \cdot x - 10 \cdot r) \cdot v / 4 + v^2 / 2 + (x - r) \cdot (2 \cdot r - 2 \cdot x - 3 \cdot v) / 2 + v \cdot (2 \cdot r - 2 \cdot x - 3 \cdot v) / 2 \leq (0::\text{real})$  (is ?t1 + ?t2 + ?t3 + ?t4  $\leq 0$ )

proof—

have ?t1 =  $5 * (x - r) * v / 2$

by auto

moreover have ?t3 =  $-((x - r)^2) - 3 * v * (x - r) / 2$

by (auto simp: field-simps power2-diff power2-eq-square)

moreover have ?t4 =  $-2 * (x - r) * v / 2 - 3 * v^2 / 2$

by (auto simp: field-simps power2-diff power2-eq-square)

ultimately have ?t1 + ?t3 + ?t4 =  $-((x - r)^2) - 3 * v^2 / 2$

by (auto simp: field-simps)

hence ?t1 + ?t2 + ?t3 + ?t4 =  $-((x - r)^2) - v^2$

by auto

also have  $\dots \leq 0$

by auto

finally show ?thesis .

qed

lemma  $c > 0 \implies Kp = 2 \implies Kd = 3 \implies [\lambda s::\text{real}^2. (5/4) * (s\$1 - xr)^2 + (s\$1 - xr) * (s\$2) / 2 +$

$(s\$2)^2 / 4 < c] \leq \text{wp}$

( $x' = (\lambda s. \chi \text{ i. if } i=1 \text{ then } s\$2 \text{ else } -Kp * (s\$1 - xr) - Kd * (s\$2)) \& G$ )



```

 $\lceil \lambda s. (5/4) * (s\$1 - xr) ^ 2 + (s\$1 - xr) * (s\$2) / 2 + (s\$2) ^ 2 / 4 < c \rceil$ 
apply(simp, rule-tac  $\mu' = \lambda s. 0$  and  $\nu' = \lambda s. 10 * (s\$1 - xr) * (s\$2) / 4 + (s\$2 ^ 2) / 2 +$ 
 $(s\$1 - xr) * (-Kp * (s\$1 - xr) - Kd * (s\$2)) / 2 + (s\$2) * (-Kp * (s\$1 - xr) - Kd * (s\$2)) / 2$  in diff-invariant-rules(3);

  clarsimp simp: forall-2 STExample9a-arith)
apply(intro poly-derivatives; (rule poly-derivatives)?)
by force+ (auto simp: field-simps power2-eq-square)

```

### STTT Tutorial: Example 9b

**lemma** wp-assign-wp:  $wp\ (x ::= e)\ (wp\ R\ [Q]) = \lceil \lambda s. \exists y. (vec\text{-}upd\ s\ x\ (e\ s),\ y) \in (wp\ R\ [Q]) \rceil$   
**by** (clarsimp simp: vec-upd-def assign-def wp-rel)

**lemma** wp-test-wp:  $wp\ [P]\ (wp\ R\ [Q]) = \lceil \lambda s. P\ s \longrightarrow (s, s) \in wp\ R\ [Q] \rceil$   
**unfolding** wp-rel **by** (clarsimp simp: p2r-def)

**lemma** in-wp-g-odeI:  
**assumes**  $(x, x) \in [I]\ [I] \subseteq wp\ (x' = f \ \&\ G\ on\ (\lambda s. T)\ S\ @\ t_0)\ [I]$   
**and**  $\lceil \lambda s. I\ s \wedge G\ s \rceil \subseteq [Q]\ y = x$   
**shows**  $(x, y) \in wp\ (x' = f \ \&\ G\ on\ (\lambda s. T)\ S\ @\ t_0)\ DINV\ I\ [Q]$   
**using** wp-g-odei[OF - assms(2,3), of I] assms(1,4) **by** (clarsimp simp: subset-eq p2r-def)

**lemma** in-wp-g-odeE:  
 $(x, y) \in wp\ (x' = f \ \&\ G\ on\ (\lambda s. T)\ S\ @\ t_0)\ DINV\ I \implies I\ x \implies [I] \subseteq wp\ (x' = f \ \&\ G\ on\ (\lambda s. T)\ S\ @\ t_0)$   
 $[I] \implies I\ y$   
**apply**(simp add: g-ode-inv-def diff-invariant-eq)  
**by** (clarsimp simp: g-ode-def g-orbital-def g-orbit-def ivp-sols-def)

**lemma** local-flow-STTT-Ex9b:

```

  local-flow ( $\lambda s::real^4. \chi\ i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } -2 * (s\$1 - s\$3) - 3 * (s\$2) \text{ else } 0)$ )
  UNIV UNIV ( $\lambda t\ s. \chi\ i. \text{if } i=1 \text{ then } \exp\ ((-2) * t) * (s\$3 - 2 * (\exp\ t) * s\$3 + (\exp\ (2 * t)) * s\$3 -$ 
 $s\$2 + (\exp\ t) * (s\$2) - s\$1 + 2 * (\exp\ t) * (s\$1))$ 
   $\text{else } (\text{if } i=2 \text{ then } (\exp\ (-2 * t)) * (-2 * s\$3 + 2 * (\exp\ t) * s\$3 + 2 * s\$2 - (\exp\ t) * s\$2 + 2 * s\$1$ 
 $- 2 * (\exp\ t) * s\$1) \text{ else } s\$i)$ )
apply(unfold-locales, simp-add: forall-4 vec-eq-iff, safe; (rule has-vderiv-on-const)?)
apply(simp add: local-lipschitz-def lipschitz-on-def dist-norm norm-vec-def L2-set-def)
unfolding UNIV-4 apply clarsimp
subgoal sorry
apply(intro poly-derivatives, rule poly-derivatives, rule poly-derivatives, force, force, force)
  apply(rule poly-derivatives, rule poly-derivatives, rule poly-derivatives, force, force)
  apply(rule poly-derivatives, force, force, rule poly-derivatives, rule poly-derivatives, force,
force)
  apply(rule poly-derivatives, force, force, rule poly-derivatives, force)
  apply(rule poly-derivatives, force, rule poly-derivatives, force, force, rule poly-derivatives)
  apply(force, rule poly-derivatives, rule poly-derivatives, force, force, rule poly-derivatives)
  apply(force, force, force simp: field-simps)
apply(intro poly-derivatives, rule poly-derivatives, rule poly-derivatives, force, force, force)
  apply(rule poly-derivatives, rule poly-derivatives, force, force)
  apply(rule poly-derivatives, force, rule poly-derivatives, rule poly-derivatives)
  apply(force, force, rule poly-derivatives, rule poly-derivatives, force, force)
  apply(rule poly-derivatives, force, rule poly-derivatives, force, force)
  apply(rule poly-derivatives, rule poly-derivatives, force, force)+
  apply(rule poly-derivatives, force, force)
by (auto simp: field-simps exp-minus-inverse)

```

**lemma**

```

assumes  $Kp = 2\ Kd = 3$ 
shows  $\lceil \lambda s::real^4. s\$2 \geq 0 \wedge s\$3 \leq s\$1 \wedge s\$1 \leq S \wedge s\$4 = (s\$3 + S) / 2$ 
 $\wedge (5/4) * (s\$1 - s\$4) ^ 2 + (s\$1 - s\$4) * (s\$2) / 2 + (s\$2) ^ 2 / 4 < ((S - s\$3) / 2) ^ 2 \rceil \leq wp$ 

```

```

(LOOP (( $\beta ::= (\lambda s. s\$1)$ );( $\delta ::= (\lambda s. (s\$3 + S)/2$ )));
 $\lceil \lambda s. (5/4) * (s\$1 - s\$4)^2 + (s\$1 - s\$4) * (s\$2)/2 + (s\$2)^2/4 < ((S - s\$3)/2)^2 \rceil \cup \lceil \lambda s. \text{True} \rceil$ );
( $x' = (\lambda t s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } -Kp * (s\$1 - s\$3) - Kd * (s\$2) \text{ else } 0)) \ \& \ (\lambda s. s\$2 \geq 0)$ ) &
 $(\lambda s. s\$2 \geq 0)$  on  $(\lambda s. \{0..\})$  UNIV @ 0 DINV  $(\lambda s. s\$3 \leq s\$1 \wedge$ 
 $(5/4) * (s\$1 - (s\$3 + S)/2)^2 + (s\$1 - (s\$3 + S)/2) * (s\$2)/2 + s\$2^2/4 < ((S - s\$3)/2)^2)$ )
INV  $(\lambda s. s\$2 \geq 0 \wedge s\$3 \leq s\$1 \wedge s\$4 = (s\$3 + S)/2 \wedge$ 
 $(5/4) * (s\$1 - s\$4)^2 + (s\$1 - s\$4) * (s\$2)/2 + (s\$2)^2/4 < ((S - s\$3)/2)^2)$ )
 $\lceil \lambda s. s\$1 \leq S \rceil$ 
unfolding assms
apply(rule wp-loopI, simp-all add: rel-aka.fbox-add2 local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex9b]

    g-ode-inv-def wp-assign-wp wp-test-wp)
apply(auto simp: field-simps)
oops

```

## STTT Tutorial: Example 10

### LICS: Example 1 Continuous car accelerates forward

```

lemma  $\lceil \lambda s::\text{real}^3. s\$2 \geq 0 \wedge s\$3 \geq 0 \rceil \leq wp$ 
 $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \geq 0))$ 
 $\lceil \lambda s. s\$2 \geq 0 \rceil$ 
by (simp-all add: local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2])

```

### LICS: Example 2 Single car drives forward

```

lemma  $A \geq 0 \implies b > 0 \implies \lceil \lambda s::\text{real}^3. s\$2 \geq 0 \rceil \leq wp$ 
(LOOP
 $((\beta ::= (\lambda s. A)) \cup (\beta ::= (\lambda s. -b)))$ ;
 $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \geq 0))$ )
INV  $(\lambda s. s\$2 \geq 0)$ 
 $\lceil \lambda s. s\$2 \geq 0 \rceil$ 
by (rule wp-loopI) (simp-all add: local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2] rel-aka.fbox-add2)

```

### LICS: Example 3a event-triggered car drives forward

```

lemma  $A \geq 0 \implies b > 0 \implies \lceil \lambda s::\text{real}^3. s\$2 \geq 0 \rceil \leq wp$ 
(LOOP
 $((\lceil \lambda s. m - s\$1 \geq 2 \rceil; (\beta ::= (\lambda s. A))) \cup (\beta ::= (\lambda s. -b)))$ ;
 $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } s\$3 \text{ else } 0)) \ \& \ (\lambda s. s\$2 \geq 0))$ )
INV  $(\lambda s. s\$2 \geq 0)$ 
 $\lceil \lambda s. s\$2 \geq 0 \rceil$ 
by (rule wp-loopI) (simp-all add: local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex2] rel-aka.fbox-add2)

```

### LICS: Example 4a safe stopping of time-triggered car

```

lemma LICSexample4a-arith:
assumes  $(0::\text{real}) \leq A < b \ v^2 \leq 2 \cdot b \cdot (m - x) \ 0 \leq t$ 
and guard:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow 0 \leq A \cdot \tau + v \wedge \tau \leq \varepsilon$ 
and key:  $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$  (is ?expr1  $\leq -$ )
shows  $(A \cdot t + v)^2 \leq 2 \cdot b \cdot (m - (A \cdot t^2 / 2 + v \cdot t + x))$ 
proof–
have  $t \leq \varepsilon \ 0 \leq v$ 
using guard  $\langle 0 \leq t \rangle$  by (force, erule-tac x=0 in allE, auto)
hence  $A \cdot t^2 + 2 \cdot t \cdot v \leq A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v$ 
using  $\langle 0 \leq A \rangle \langle 0 \leq t \rangle$ 
by (smt mult-less-cancel-left-disj mult-right-mono power-less-imp-less-base)
hence  $v^2 + (A + b) \cdot (A \cdot t^2 + 2 \cdot t \cdot v) \leq v^2 + (A + b) \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)$ 
using  $\langle 0 \leq A \rangle \langle 0 < b \rangle$  by (smt mult-left-mono)
also have ... = ?expr1
by auto

```

finally have  $v^2 + (A + b) \cdot (A \cdot t^2 + 2 \cdot t \cdot v) \leq 2 \cdot b \cdot (m - x)$   
 using key by auto  
 thus ?thesis  
 by (auto simp: field-simps power2-eq-square)  
 qed

**lemma**  $A \geq 0 \implies b > 0 \implies \lceil \lambda s :: \text{real}^4. s\$2^2 \leq 2*b*(m-s\$1) \wedge s\$2 \geq 0 \rceil \leq wp$   
 (LOOP  
 ((( $\lceil \lambda s. 2*b*(m-s\$1) \geq s\$2^2 + (A+b)*(A*\varepsilon^2 + 2*\varepsilon*(s\$2)) \rceil$ ); ( $\mathcal{I} ::= (\lambda s. A)$ )  $\cup (\mathcal{I} ::= (\lambda s. -b)$ ));  
 ( $\mathcal{I} ::= (\lambda s. 0)$ );  
 ( $x' = (\lambda s. \chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1)) \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)$ ))  
 INV ( $\lambda s. s\$2^2 \leq 2*b*(m-s\$1)$ ))  
 $\lceil \lambda s. s\$1 \leq m \rceil$   
 apply (rule wp-loopI)  
 apply (simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-STTT-Ex5])  
 apply (safe, smt not-sum-power2-lt-zero zero-compare-simps(10))  
 using LICExample4a-arith[of  $A \ b \ m \ -\varepsilon$ ] apply force  
 by (auto simp: power2-diff power2-eq-square[symmetric] algebra-simps(18,19)  
 mult.assoc[symmetric] power-mult-distrib)

### LICS: Example 4b progress of time-triggered car

notation *rel-aka.fdia* ( $\diamond$ )

**lemma** *in-fdia-iff-wp*:  $(s, s) \in \diamond R \lceil P \rceil \longleftrightarrow (s, s) \in \text{rel-ad } (wp \ R \ (\text{rel-ad } \lceil P \rceil))$   
 unfolding *rel-ad-def* *rel-aka.fdia-def* *rel-aka.fbox-def* by (auto simp: *p2r-def*)

**lemma**  $\varepsilon > (0 :: \text{real}) \implies A > 0 \implies b > 0 \implies$   
 $\forall p. \exists m. (s, s) \in \diamond ($   
 LOOP ( $\lceil \lambda s. 2*b*(m-x) \geq s\$2^2 + (A+b)*A*\varepsilon^2 + 2*\varepsilon*v \rceil$ ); ( $\mathcal{I} ::= (\lambda s. A)$ )  $\cup (\mathcal{I} ::= (\lambda s. -b)$ );  
 ( $\mathcal{I} ::= (\lambda s. 0)$ );  
 ( $x' = (\lambda s. f \ 1 \ (s\$3) \ s) \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)$ )  
 INV ( $\lambda s. \text{True}$ )  $\lceil \lambda s. s\$1 \geq p \rceil$   
 apply (subst *in-fdia-iff-wp*, *simp*)  
 apply (clarsimp *simp: rel-ad-def*)  
 apply (intro *exI conjI allI*)  
 apply clarsimp  
 oops

### LICS: Example 4c relative safety of time-triggered car

**lemma** *in-wp-loopI*:  
 $I \ x \implies \lceil I \rceil \subseteq \lceil Q \rceil \implies \lceil I \rceil \subseteq wp \ R \ \lceil I \rceil \implies y = x \implies (x, y) \in wp \ (LOOP \ R \ INV \ I) \ \lceil Q \rceil$   
 using *wp-loopI*[of  $I \ I \ Q \ R$ ] apply *simp*  
 apply (subgoal-tac  $(x, x) \in \lceil I \rceil$ )  
 by (simp add: *subset-eq*, *simp add: p2r-def*)

**lemma** (in *local-flow*) *in-wp-g-ode-subset*:  
 assumes  $\bigwedge s. s \in S \implies 0 \in U \ s \wedge \text{is-interval } (U \ s) \wedge U \ s \subseteq T$   
 shows  $(s, s) \in wp \ (x' = (\lambda t. f) \ \& \ G \text{ on } U \ S \ @ \ 0) \lceil Q \rceil \longleftrightarrow (s \in S \longrightarrow (\forall t \in U \ s. (\forall \tau. \tau \in U \ s \wedge \tau \leq t \longrightarrow G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$   
 by (subst *wp-g-ode-subset*[OF *assms*], *simp-all add: p2r-def*)

**abbreviation** *LICS-Ex4c-f* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real}^4 \Rightarrow \text{real}^4 \ (f)$   
 where  $f \text{ time acc } s \equiv (\chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then acc else if } i=3 \text{ then } 0 \text{ else time)})$

**lemma** *local-flow-LICS-Ex4c-1*:

*local-flow* (*f k a*) *UNIV UNIV*  
 $(\lambda t s. \chi i. \text{if } i=1 \text{ then } a * t^2/2 + s\$2 * t + s\$1 \text{ else}$   
 $\quad (\text{if } i=2 \text{ then } a * t + s\$2 \quad \text{else}$   
 $\quad (\text{if } i=3 \text{ then } s\$3 \quad \text{else}$   
 $\quad \quad k * t + s\$4 \quad \quad \quad ))))$   
**apply**(*unfold-locales*, *simp-all add: local-lipschitz-def forall-2 lipschitz-on-def*)  
**apply**(*clarsimp*, *rule-tac x=1 in exI*) +  
**apply**(*clarsimp simp: dist-norm norm-vec-def L2-set-def*)  
**unfolding** *UNIV-4* **by** (*auto intro!: poly-derivatives simp: forall-4 vec-eq-iff*)

**lemma** *local-flow-LICS-Ex4c-2*:

*local-flow* ( $\lambda s. f k (s\$3) s$ ) *UNIV UNIV*  
 $(\lambda t s. \chi i. \text{if } i=1 \text{ then } s\$3 * t^2/2 + s\$2 * t + s\$1 \text{ else}$   
 $\quad (\text{if } i=2 \text{ then } s\$3 * t + s\$2 \quad \text{else}$   
 $\quad (\text{if } i=3 \text{ then } s\$3 \quad \text{else}$   
 $\quad \quad k * t + s\$4 \quad \quad \quad ))))$   
**apply**(*unfold-locales*, *simp-all add: local-lipschitz-def forall-2 lipschitz-on-def*)  
**apply**(*clarsimp*, *rule-tac x=1 in exI*) +  
**apply**(*clarsimp simp: dist-norm norm-vec-def L2-set-def*)  
**unfolding** *UNIV-4* **by** (*auto intro!: poly-derivatives simp: forall-4 vec-eq-iff*)

**lemma** *LICSexample4c-arith1*:

**assumes**  $v^2 \leq 2 \cdot b \cdot (m - x) \ 0 \leq t \ A \geq 0 \ b > 0$   
**and** *key*:  $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$   
**and** *guard*:  $\forall \tau. \ 0 \leq \tau \wedge \tau \leq t \longrightarrow (0::\text{real}) \leq A \cdot \tau + v \wedge \tau \leq \varepsilon$   
**shows**  $(A \cdot t + v)^2 \leq 2 \cdot b \cdot (m - (A \cdot t^2 / 2 + v \cdot t + x))$  (**is - ≤ ?rhs**)

**proof**—

**have**  $t \leq \varepsilon \ 0 \leq \varepsilon \ 0 \leq v$   
**using** *guard*  $\langle 0 \leq t \rangle$  **by** (*force*, *erule-tac x=0 in allE*, *simp*, *erule-tac x=0 in allE*, *simp*)  
**hence** *obs1*:  $A \cdot t^2 + 2 \cdot t \cdot v \leq A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v$   
**using**  $\langle A \geq 0 \rangle \langle 0 \leq t \rangle \langle t \leq \varepsilon \rangle$  **by** (*smt mult-mono power-mono zero-compare-simps*(12))  
**have** *obs2*:  $?rhs + A * b * t^2 + 2 * b * v * t = 2 * b * (m - x)$   
**by** (*simp add: field-simps*)  
**have**  $(A \cdot t + v)^2 + A * b * t^2 + 2 * b * v * t = v^2 + (A \cdot (A \cdot t^2 + 2 \cdot t \cdot v) + b \cdot (A \cdot t^2 + 2 \cdot t \cdot v))$   
**by** (*simp add: field-simps power2-eq-square*)  
**also** **have**  $\dots \leq v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v))$   
**using** *obs1*  $\langle A \geq 0 \rangle \langle b > 0 \rangle$  **by** (*smt mult-less-cancel-left*)  
**also** **have**  $\dots \leq 2 * b * (m - x)$   
**using** *key* .  
**finally** **show** *?thesis*  
**using** *obs2* **by** *auto*

**qed**

**lemma**

**assumes**  $A \geq 0 \ b > 0 \ s\$2 \geq 0$   
**shows**  $(s, s) \in wp \ (x' = (f \ 0 \ (-b))) \ \& \ (\lambda s. \ \text{True})) \ [\lambda s. \ s\$1 \leq m] \Longrightarrow$   
 $(s, s) \in wp$   
 $(LOOP$   
 $\quad (([\lambda s. \ 2 * b * (m - s\$1) \geq s\$2^2 + (A + b) * (A * \varepsilon^2 + 2 * \varepsilon * (s\$2))]; (\mathcal{I} ::= (\lambda s. \ A))) \cup (\mathcal{I} ::= (\lambda s. \ -b)))$   
 $\quad (\mathcal{I} ::= (\lambda s. \ 0)))$   
 $\quad (x' = (\lambda s. \ f \ 1 \ (s\$3) \ s) \ \& \ (\lambda s. \ s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)))$   
 $INV \ (\lambda s. \ s\$2^2 \leq 2 * b * (m - s\$1))) \ [\lambda s. \ s\$1 \leq m]$   
**apply**(*subst (asm) local-flow.in-wp-g-ode-subset[OF local-flow-LICS-Ex4c-1]*, *simp-all*)  
**apply**(*rule in-wp-loopI*)  
**apply**(*erule-tac x=s\\$2/b in allE*)  
**using**  $\langle b > 0 \rangle \langle s\$2 \geq 0 \rangle$  **apply**(*simp add: field-simps power2-eq-square*, *simp*)

```

  apply (smt ⟨b > 0⟩ mult-sign-intros(6) sum-power2-ge-zero)
apply(simp add: rel-aka.fbox-add2)
  apply(simp-all add: local-flow.wp-g-ode-subset[OF local-flow-LICS-Ex4c-2], safe)
using LICSexample4c-arith1[OF - - ⟨0 ≤ A⟩ ⟨0 < b⟩] apply force
by (auto simp: field-simps power2-eq-square)

```

no-notation LICS-Ex4c-f (f)

### LICS: Example 5 Controllability Equivalence

lemma LICSexample5-arith1:

```

  assumes (0::real) < b 0 ≤ t
    and key: v2 ≤ 2 · b · (m - x)
  shows v · t - b · t2 / 2 + x ≤ m

```

proof—

```

  have v2 ≤ 2 · b · (m - x) + (b · t - v)2
    using key by (simp add: add-increasing2)
  hence b2 * t2 - 2 * b * v * t ≥ 2 * b * x - 2 * b * m
    by (auto simp: field-simps power2-diff)
  hence (b2/b) * t2 - 2 * (b/b) * v * t ≥ 2 * (b/b) * x - 2 * (b/b) * m
    using ⟨b > 0⟩ by (auto simp: field-simps)
  thus ?thesis
    using ⟨b > 0⟩ by (simp add: power2-eq-square)

```

qed

lemma LICSexample5-arith2:

```

  assumes (0::real) < b 0 ≤ v ∀ t ∈ {0..}. v · t - b · t2 / 2 + x ≤ m
  shows v2 ≤ 2 · b · (m - x)

```

proof(cases v = 0)

```

  case True
  have m - x ≥ 0
    using assms by (erule-tac x=0 in ballE, auto)
  thus ?thesis
    using assms True by auto

```

next

```

  case False
  hence obs: v > 0 ∧ (∃ k. k > 0 ∧ v = b * k)
    using assms(1,2) by (metis (no-types, hide-lams) divide-pos-pos divide-self-if
      less-eq-real-def linorder-not-le mult-1-right mult-1s(1) times-divide-eq-left)
  {fix t::real assume t ≥ 0
    hence v · t - b · t2 / 2 + x ≤ m
      using assms by auto
    hence - (b2) * t2 + 2 * b * v * t ≤ 2 * b * m - 2 * b * x
      using ⟨b > 0⟩ apply(simp add: field-simps)
    by (metis (no-types, hide-lams) Groups.mult-ac(1) nat-distrib(2) power2-eq-square real-mult-le-cancel-iff2)
    hence v2 ≤ 2 * b * (m - x) + (b2 * t2 + v2 - 2 * b * v * t)
      by (simp add: field-simps)
    also have ... = 2 * b * (m - x) + (b * t - v)2
      by (simp add: power2-diff power-mult-distrib)
    finally have v2 ≤ 2 * b * (m - x) + (b * t - v)2 .}
  hence ∀ t ≥ 0. v2 ≤ 2 · b · (m - x) + (b · t - v)2
    by blast
  then obtain k where v2 ≤ 2 · b · (m - x) + (b · k - v)2 ∧ k > 0 ∧ v = b * k
    using obs by fastforce
  then show ?thesis
    by auto

```

qed

lemma b > 0 ⇒ s\$2 ≥ 0 ⇒

$(s,s) \in [\lambda s::\text{real}^2. s\$2^2 \leq 2*b*(m-s\$1)] \longleftrightarrow (s,s) \in wp$   
 $(x' = (\lambda s. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } -b) \ \& \ (\lambda s. \text{True}))$   
 $[\lambda s. s\$1 \leq m]$   
**apply**(subst local-flow.wp-g-ode-subset[where  $T=UNIV$   
and  $\varphi = \lambda t s. \chi i::2. \text{if } i=1 \text{ then } -b * t^2/2 + s\$2 * t + s\$1 \text{ else } -b * t + s\$2]$ )  
**apply**(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)  
**apply**(clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac  $x=1$  in exI)+  
**unfolding** UNIV-2 **apply** clarsimp  
**apply**(force intro!: poly-derivatives)  
**using** exhaust-2 **apply**(force simp: vec-eq-iff)  
**by** (auto simp: p2r-def LICSexample5-arith1 LICSexample5-arith2)

### LICS: Example 6 MPC Acceleration Equivalence

**lemma** LICSexample6-arith1:

**assumes**  $0 \leq v$   $0 < b$   $0 \leq A$   $0 \leq \varepsilon$  **and guard:**  $\forall t \in \{0..\}. (\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon) \longrightarrow (\forall \tau \in \{0..\}).$

$A \cdot t \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot t^2 / 2 + v \cdot t + x) \leq (m::\text{real})$   
**shows**  $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$

**proof**—

**{fix**  $\tau::\text{real}$   
**assume**  $\tau \geq 0$   
**hence**  $A \cdot \varepsilon \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + v \cdot \varepsilon + x) \leq m$   
**using** guard  $\langle 0 \leq \varepsilon \rangle$  **apply**(erule-tac  $x=\varepsilon$  in ballE)  
**by** (erule impE, auto simp: closed-segment-eq-real-ivl)  
**hence**  $2 * (A \cdot \varepsilon \cdot \tau + v \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + v \cdot \varepsilon + x)) * b \leq 2 * m * b$   
**using**  $\langle 0 < b \rangle$  **by** (meson less-eq-real-def mult-left-mono mult-right-mono rel-simps(51))  
**hence**  $2 * A \cdot \varepsilon \cdot \tau \cdot b + 2 * v \cdot \tau \cdot b - b^2 \cdot \tau^2 + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$   
**using**  $\langle 0 < b \rangle$  **apply**(simp add: algebra-simps(17,18,19,20) add.assoc[symmetric]  
power2-eq-square[symmetric] mult.assoc[symmetric])  
**by** (simp add: mult.commute mult.left-commute power2-eq-square)}  
**hence**  $\forall \tau \geq 0. 2 * A \cdot \varepsilon \cdot \tau \cdot b + 2 * v \cdot \tau \cdot b - b^2 \cdot \tau^2 + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$   
**by** blast  
**moreover have**  $2 * A \cdot \varepsilon \cdot ((A \cdot \varepsilon + v)/b) \cdot b + 2 * v \cdot ((A \cdot \varepsilon + v)/b) \cdot b - b^2 \cdot ((A \cdot \varepsilon + v)/b)^2 =$   
 $2 * A \cdot \varepsilon \cdot (A \cdot \varepsilon + v) + 2 * v \cdot (A \cdot \varepsilon + v) - (A \cdot \varepsilon + v)^2$   
**using**  $\langle 0 < b \rangle$  **by** (simp add: field-simps)  
**moreover have**  $\dots = v^2 + A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)$   
**using**  $\langle 0 < b \rangle$  **by** (simp add: field-simps power2-eq-square)  
**moreover have**  $(A \cdot \varepsilon + v)/b \geq 0$   
**using** assms **by** auto  
**ultimately have**  $v^2 + A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b * (A \cdot \varepsilon^2 + 2 * v \cdot \varepsilon) \leq 2 * b * (m - x)$   
**using** assms **by** (erule-tac  $x=(A \cdot \varepsilon + v)/b$  in allE, auto)  
**thus** ?thesis  
**by** argo  
**qed**

**lemma** LICSexample6-arith2:

**assumes**  $0 \leq v$   $0 < b$   $0 \leq A$   $0 \leq t$   $0 \leq \tau$   $t \leq \varepsilon$   
**and**  $v^2 + (A \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v) + b \cdot (A \cdot \varepsilon^2 + 2 \cdot \varepsilon \cdot v)) \leq 2 \cdot b \cdot (m - x)$   
**shows**  $A \cdot \varepsilon \cdot \tau + s \$ 2 \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + s \$ 2 \cdot \varepsilon + s \$ 1) \leq m$

**sorry**

**lemma** local-flow-LICS-Ex6:

local-flow  $(\lambda s::\text{real}^3. \chi i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } k1 \text{ else } k2)) \text{ UNIV UNIV}$   
 $(\lambda t s. \chi i. \text{if } i = 1 \text{ then } k1 * t^2/2 + s\$2 * t + s\$1 \text{ else } (\text{if } i=2 \text{ then } k1 * t + s\$2 \text{ else } k2 * t + s\$3))$   
**apply**(unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def)  
**apply**(clarsimp, rule-tac  $x=1$  in exI)+  
**apply**(clarsimp simp: dist-norm norm-vec-def L2-set-def)

**unfolding** *UNIV-3* by (auto intro!: poly-derivatives simp: forall-3 vec-eq-iff)

**lemma**  $s\$2 \geq 0 \implies b > 0 \implies A \geq 0 \implies \varepsilon \geq 0 \implies$   
 $(s, s) \in wp ((\mathcal{I} ::= (\lambda s. 0)));$   
 $(x' = (\lambda s :: real^3. \chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } A \text{ else } 1)) \ \& \ (\lambda s. s\$3 \leq \varepsilon)))$   
 $(wp (x' = (\lambda s. \chi \ i. \text{if } i=1 \text{ then } s\$2 \text{ else } (\text{if } i=2 \text{ then } -b \text{ else } 0)) \ \& \ (\lambda s. True)) \ [\lambda s. s\$1 \leq m])$   
 $\longleftrightarrow$   
 $(s, s) \in [\lambda s. 2*b*(m-s\$1) \geq s\$2^2 + (A+b)*(A*\varepsilon^2 + 2*\varepsilon*(s\$2))]$   
**apply** (simp add: le-wp-choice-iff local-flow.wp-g-ode-subset[OF local-flow-LICS-Ex6], safe)  
**apply** (force simp: LICSexample6-arith1)  
**apply** (erule-tac  $x=t$  in allE; clarsimp)  
**apply** (rename-tac  $t \ \tau$ )  
**apply** (rule-tac  $b=A \cdot \varepsilon \cdot \tau + s \ \$ \ 2 \cdot \tau - b \cdot \tau^2 / 2 + (A \cdot \varepsilon^2 / 2 + s \ \$ \ 2 \cdot \varepsilon + s \ \$ \ 1)$  in order.trans)  
**apply** (smt divide-le-cancel mult-left-mono mult-right-mono power2-less-imp-less)  
by (auto simp: LICSexample6-arith2)

## LICS: Example 7 Model-Predictive Control Design Car

**notation** *LICS-Ex4c-f* ( $f$ )

**lemma**  $s\$2 \geq 0 \implies b > 0 \implies A \geq 0 \implies$   
 $(s, s) \in (wp (x' = (f \ 0 \ (-b)) \ \& \ (\lambda s. True))) \ [\lambda s. s\$1 \leq m] \implies$   
 $(s, s) \in wp ($   
 $LOOP$   
 $(([\lambda s. (s, s) \in wp ((\mathcal{I} ::= (\lambda s. 0)); (x' = (f \ 1 \ A) \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon))$   
 $(wp (x' = (f \ 0 \ (-b)) \ \& \ (\lambda s. True)) \ [\lambda s. s\$1 \leq m])]; (\mathcal{I} ::= (\lambda s. A))) \cup (\mathcal{I} ::= (\lambda s. -b))));$   
 $(\mathcal{I} ::= (\lambda s. 0)); (x' = (\lambda s. f \ 1 \ (s\$3 \ s) \ \& \ (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)))$   
 $INV \ (\lambda s. (s, s) \in wp (x' = (f \ 0 \ (-b)) \ \& \ (\lambda s. True)) \ [\lambda s. s\$1 \leq m])) \ [\lambda s. s\$1 \leq m]$   
**apply** (rule in-wp-loopI, simp-all add: local-flow.wp-g-ode-subset[OF local-flow-LICS-Ex4c-1]  
 $local-flow.wp-g-ode-subset[OF local-flow-LICS-Ex4c-2] \ le-wp-choice-iff, \ safe)$   
**apply** (erule-tac  $x=0$  in allE, erule-tac  $x=0$  in allE, simp)  
**apply** (thin-tac  $\forall t \geq 0. s \ \$ \ 2 \cdot t - b \cdot t^2 / 2 + s \ \$ \ 1 \leq m$ ) defer  
**apply** (thin-tac  $\forall t \geq 0. s \ \$ \ 2 \cdot t - b \cdot t^2 / 2 + s \ \$ \ 1 \leq m$ )  
**apply** (rename-tac  $x \ t \ \tau$ )  
subgoal sorry  
subgoal sorry  
done

**no-notation** *LICS-Ex4c-f* ( $f$ )

### 0.16.2 Advanced

#### ETCS: Essentials

**locale** *ETCS* =  
**fixes**  $\varepsilon \ b \ A \ m :: real$

**begin**

**abbreviation**  $stopDist \ v \equiv v^2 / (2*b)$

**abbreviation**  $accCompensation \ v \equiv ((A/b) + 1) \cdot ((A/2) \cdot \varepsilon^2 + \varepsilon \cdot v)$

**abbreviation**  $SB \ v \equiv stopDist \ v + accCompensation \ v$

**abbreviation**  $initial\ m'\ z\ v \equiv (v \geq 0 \wedge (m' - z) \geq stopDist\ v)$

**abbreviation**  $safe\ m'\ z\ v\ \delta \equiv z \geq m' \longrightarrow v \leq \delta$

**abbreviation**  $loopInv\ m'\ z\ v \equiv v \geq 0 \wedge m' - z \geq stopDist\ v$

**abbreviation**  $ctrl \equiv [\lambda s::real^4. m - s\$1 \leq SB\ (s\$2)];(3 ::= (\lambda s. -b)) \cup$   
 $[\lambda s. m - s\$1 \geq SB\ (s\$2)];(3 ::= (\lambda s. A))$

**abbreviation**  $drive \equiv (4 ::= (\lambda s. 0));$   
 $(x' = (\lambda s::real^4. \chi\ i. \text{ if } i=1 \text{ then } s\$2 \text{ else (if } i=2 \text{ then } s\$3 \text{ else (if } i=3 \text{ then } 0 \text{ else } 1))))$   
 $\& (\lambda s. s\$2 \geq 0 \wedge s\$4 \leq \varepsilon)$

**lemma** *ETCS-arith1*:

**assumes**  $0 < b\ 0 \leq A\ 0 \leq v\ 0 \leq t$   
**and**  $v^2 / (2 \cdot b) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) / b + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v)) \leq m - x$  (**is**  $?expr1 \leq m - x$ )  
**and guard**:  $\forall \tau. 0 \leq \tau \wedge \tau \leq t \longrightarrow \tau \leq \varepsilon$   
**shows**  $(A \cdot t + v)^2 / (2 \cdot b) \leq m - (A \cdot t^2 / 2 + v \cdot t + x)$  (**is**  $?lhs \leq ?rhs$ )

**proof**—

**have**  $2 \cdot b \cdot (v^2 / (2 \cdot b) + (A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) / b + (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v))) \leq 2 \cdot b \cdot (m - x)$  (**is**  $?expr2 \leq 2 \cdot b \cdot (m - x)$ )  
**using**  $\langle 0 < b \rangle$  *mult-left-mono*[*OF*  $\langle ?expr1 \leq m - x \rangle$ , *of*  $2 \cdot b$ ] **by** *auto*  
**also have**  $?expr2 = v^2 + 2 \cdot A \cdot (A \cdot \varepsilon^2 / 2 + \varepsilon \cdot v) + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v$   
**using**  $\langle 0 < b \rangle$  **by** (*auto simp: field-simps*)  
**also have**  $\dots = v^2 + A^2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v$   
**by** (*auto simp: field-simps power2-eq-square*)  
**finally have**  $obs: v^2 + A^2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2 + 2 \cdot b \cdot \varepsilon \cdot v \leq 2 \cdot b \cdot (m - x)$  (**is**  $?expr3\ \varepsilon \leq 2 \cdot b \cdot (m - x)$ ) .  
**have**  $t \leq \varepsilon$   
**using guard**  $\langle 0 \leq t \rangle$  **by** *auto*  
**hence**  $v^2 + A^2 \cdot t^2 + b \cdot A \cdot t^2 \leq v^2 + A^2 \cdot \varepsilon^2 + b \cdot A \cdot \varepsilon^2$   
**using** *power-mono*[*OF*  $\langle t \leq \varepsilon \rangle\ \langle 0 \leq t \rangle$ , *of*  $2$ ]  
**by** (*smt assms(1,2) mult-less-cancel-left zero-compare-simps(4) zero-le-power*)  
**hence**  $v^2 + A^2 \cdot t^2 + 2 \cdot A \cdot t \cdot v + b \cdot A \cdot t^2 \leq v^2 + A^2 \cdot \varepsilon^2 + 2 \cdot A \cdot \varepsilon \cdot v + b \cdot A \cdot \varepsilon^2$   
**using** *assms(1,2,3,4)*  $\langle t \leq \varepsilon \rangle$  **by** (*smt mult-left-mono mult-right-mono*)  
**hence**  $?expr3\ t \leq 2 \cdot b \cdot (m - x)$   
**using** *assms(1,2,3,4)*  $\langle t \leq \varepsilon \rangle$  **obs** **by** (*smt mult-right-mono real-mult-le-cancel-iff2*)  
**hence**  $A^2 \cdot t^2 + v^2 + 2 \cdot A \cdot t \cdot v \leq 2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x$   
**by** (*simp add: right-diff-distrib*)  
**hence**  $(A \cdot t + v)^2 \leq 2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x$   
**unfolding** *cross3-simps(29)*[*of*  $A\ t\ 2$ ] *power2-sum*[*of*  $A \cdot t\ v$ ] **by** (*simp add: mult.assoc*)  
**hence**  $?lhs \leq (2 \cdot b \cdot m - b \cdot A \cdot t^2 - 2 \cdot b \cdot t \cdot v - 2 \cdot b \cdot x) / (2 \cdot b)$  (**is**  $- \leq ?expr4$ )  
**using**  $\langle 0 < b \rangle$  *divide-right-mono* **by** *fastforce*  
**also have**  $?expr4 = ?rhs$   
**using**  $\langle 0 < b \rangle$  **by** (*auto simp: field-simps*)  
**finally show**  $?lhs \leq ?rhs$  .

**qed**

**lemma**  $b > 0 \implies A \geq 0 \implies \varepsilon \geq 0 \implies$

$[\lambda s. initial\ m\ (s\$1)\ (s\$2)] \leq wp\ (LOOP\ ctrl; drive\ INV\ (\lambda s. loopInv\ m\ (s\$1)\ (s\$2)))\ [\lambda s. s\$1 \leq m]$

**apply** (*rule wp-loopI*)

**apply** (*simp-all add: le-wp-choice-iff local-flow.wp-g-ode-subset*[*OF* *local-flow-STTT-Ex5*], *safe*)

**apply** (*smt divide-le-cancel divide-minus-left not-sum-power2-lt-zero*)

**apply**(*auto simp: field-simps power2-eq-square*)[1]

**using** *ETCS-arith1* **by** *force*



### 0.16.3 ETCS: Proposition 1 (Controllability)

**abbreviation** *assumptions*  $v \delta \equiv (v \geq 0 \wedge \delta \geq 0 \wedge b > 0)$

**lemma** *ETCS-Prop1-arith1*:

**assumes**  $0 \leq v \ 0 \leq \delta \ 0 < b \ x \leq m$   
**and**  $\forall t \in \{0..\}. (\forall \tau \in \{0--t\}. b \cdot \tau \leq v) \longrightarrow$   
 $m \leq v \cdot t - b \cdot t^2 / 2 + x \longrightarrow v - b \cdot t \leq \delta$   
**shows**  $v^2 - \delta^2 \leq 2 \cdot b \cdot (m - x)$

**sorry**

**lemma** *ETCS-Prop1-arith2*:

**assumes**  $0 \leq v \ 0 \leq \delta \ 0 < b \ x \leq m \ 0 \leq t$   
**and key:**  $v^2 - \delta^2 \leq 2 \cdot b \cdot (m - x) \ m \leq v \cdot t - b \cdot t^2 / 2 + x$   
**and guard:**  $\forall \tau \in \{0--t\}. b \cdot \tau \leq v$   
**shows**  $v - b \cdot t \leq \delta$

**proof**—

**have**  $2 \cdot b \cdot (m - x) \leq 2 \cdot b \cdot (v \cdot t - b \cdot t^2 / 2) - v^2 + v^2$   
**using** *key(2) <0 < b> by simp*  
**also have**  $\dots = v^2 - (v - b \cdot t)^2$   
**using** *<0 < b> by (simp add: power2-diff field-simps power2-eq-square)*  
**finally have**  $(v - b \cdot t)^2 \leq \delta^2$   
**using** *key(1) by simp*  
**thus**  $v - b \cdot t \leq \delta$   
**using guard** *<0 ≤ t> <0 ≤ δ> by auto*

**qed**

**lemma** *assumptions*  $(s\$2) \delta \wedge (s\$1) \leq m \implies$

$(s, s) \in wp \ (x' = (\lambda t s :: real^2. \chi \ i. \text{if } i=1 \text{ then } (s\$2) \text{ else } -b) \ \& \ (\lambda s. s\$2 \geq 0) \text{ on } (\lambda s. \{0..\}) \ UNIV \ @ \ 0)$   
 $\lceil \lambda s. s\$1 \geq m \longrightarrow s\$2 \leq \delta \rceil$

$\longleftrightarrow (s, s) \in \lceil \lambda s. s\$2^2 - \delta^2 \leq 2 * b * (m - s\$1) \rceil$

**apply**(*subst local-flow.wp-g-ode-subset*[**where**  $T = UNIV$   
**and**  $\varphi = \lambda t s. \chi \ i :: 2. \text{if } i=1 \text{ then } -b * t^2 / 2 + s\$2 * t + s\$1 \text{ else } -b * t + s\$2$ ])  
**apply**(*unfold-locales, simp-all add: local-lipschitz-def forall-2 lipschitz-on-def*)  
**apply**(*clarsimp simp: dist-norm norm-vec-def L2-set-def, rule-tac x=1 in exI*) +  
**unfolding** *UNIV-2 using exhaust-2*  
**by** (*auto intro!: poly-derivatives simp: vec-eq-iff*  
*ETCS-Prop1-arith1 closed-segment-eq-real-ivl ETCS-Prop1-arith2*)

### 0.16.4 ETCS: Proposition 4 (Reactivity)

**term** *assumptions*  $v \delta$

**abbreviation** *controllable*  $m' z v \delta \equiv v^2 - \delta^2 \leq 2 * b * (m' - z) \wedge \text{assumptions } v \delta$

**term** *drive*

**lemma**  $\delta \geq 0 \wedge b > 0 \wedge \varepsilon > 0 \wedge A > 0 \wedge s\$2 \geq 0 \implies$

$(s, s) \in \lceil \lambda s. \forall m. \lceil \lambda s. m - s\$1 \geq sb \wedge \text{controllable } m \ (s\$1) \ (s\$2) \ \delta \rceil \leq wp \ ((\exists ::= (\lambda s. A)); drive) \lceil \lambda s. \text{controllable } m \ (s\$1) \ (s\$2) \ \delta \rceil$

$\longleftrightarrow sb \geq (s\$2^2 - \delta^2) / (2 * b) + (A / b + 1) * (A / 2 * \varepsilon^2 + \varepsilon * (s\$2))$

**apply** (*simp-all add: local-flow.wp-g-ode-subset*[*OF local-flow-STTT-Ex5*], *clarsimp simp: p2r-def*)

**apply**(*rule iffI, safe*)

**oops**

end

end

## 0.17 PID Examples

We prove partial correctness specifications of some hybrid systems with our verification components.

**theory** *PID-Examples*

**imports** *HS-VC-Spartan*

**begin**

**abbreviation** *time-counter-vars*  $\equiv \{ "t", "time", "counter" \}$

**typedef** *time-counter* = *time-counter-vars*

**morphisms** *to-str to-var*

**by** *blast*

**notation** *to-var* ( $\downarrow_V$ )

**lemma** *card-time-counter*:  $CARD(\text{time-counter}) = 3$

**using** *type-definition.card type-definition-time-counter* **by** *fastforce*

**instance** *time-counter::finite*

**apply**(*standard, subst bij-betw-finite[of to-str UNIV time-counter-vars]*)

**apply**(*rule bij-betwI'*)

**apply** (*simp add: to-str-inject*)

**using** *to-str* **apply** *blast*

**apply** (*metis to-var-inverse UNIV-I*)

**by** *simp*

**lemma** *time-counter-univ*:  $(UNIV::\text{time-counter set}) = \{ \downarrow_V "t", \downarrow_V "time", \downarrow_V "counter" \}$

**by** *auto (metis to-str to-str-inverse insertE singletonD)*

**lemma** *time-counter-exhaust*:  $x = \downarrow_V "t" \vee x = \downarrow_V "time" \vee x = \downarrow_V "counter"$

**using** *time-counter-univ* **by** *auto*

**lemma** *time-counter-sum*:

**fixes**  $f :: \text{time-counter} \Rightarrow ('a::\text{banach})$

**shows**  $(\sum (i::\text{time-counter}) \in UNIV. f i) = f (\downarrow_V "t") + f (\downarrow_V "time") + f (\downarrow_V "counter")$

**unfolding** *time-counter-univ* **by** (*simp add: to-var-inject*)

**abbreviation** *val-time-counter* ::  $\text{real}^{\text{time-counter}} \Rightarrow \text{string} \Rightarrow \text{real}$  (**infixl**  $\downarrow_V$  90)

**where**  $s \downarrow_V \text{string} \equiv s \$ (to-var \text{string})$

**lemma** *time-counter-induct*:  $P (\downarrow_V "t") \Longrightarrow P (\downarrow_V "time") \Longrightarrow P (\downarrow_V "counter") \Longrightarrow \forall i. P i$

**using** *time-counter-exhaust* **by** *metis*

**abbreviation** *time-counter-vec-field* ::  $\text{real}^{\text{time-counter}} \Rightarrow \text{real}^{\text{time-counter}} (f)$

**where**  $f s \equiv (\chi i. \text{if } i = \downarrow_V "t" \text{ then } 1 \text{ else } 0)$

**abbreviation** *time-counter-flow* ::  $\text{real} \Rightarrow \text{real}^{\text{time-counter}} \Rightarrow \text{real}^{\text{time-counter}} (\varphi)$

**where**  $\varphi t s \equiv (\chi i. \text{if } i = \downarrow_V "t" \text{ then } t + s \downarrow_V "t" \text{ else } s \$ i)$

**lemma** *local-lipschitz-time-counter-vec-field*: *local-lipschitz UNIV UNIV* ( $\lambda t. f$ )

**apply**(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)

```

apply(clarsimp, rule-tac x=1 in exI, clarsimp, rule-tac x=1 in exI)
apply(clarsimp simp: norm-vec-def L2-set-def)
unfolding time-counter-sum by (simp add: to-var-inject)

```

```

lemma local-flow-time-counter: local-flow f UNIV UNIV  $\varphi$ 
apply(unfold-locales, simp-all add: vec-eq-iff local-lipschitz-time-counter-vec-field)
apply(rule time-counter-induct, simp-all add: to-var-inject)
using time-counter-exhaust by (auto intro!: poly-derivatives)

```

```

lemma time-loop-verif:  $(\lambda s. s \downarrow_V''t''=0) \leq |$ 
  LOOP
    (IF  $(\lambda s. s \downarrow_V''t'' = 0.01)$  THEN
       $(\downarrow_V''t'' ::= (\lambda s. 0))$ 
    ELSE
       $(x' = f \ \& \ (\lambda s. s \downarrow_V''t'' \leq 0.01)))$ 
    INV  $(\lambda s. s \downarrow_V''t'' \leq 0.01)$ ]
 $(\lambda s. s \downarrow_V''t'' \leq 0.01)$ 
apply(rule fbox-loopI)
  apply(clarsimp simp: to-var-inject)
  apply(clarsimp simp: to-var-inject)
  apply(clarsimp simp: to-var-inject)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-time-counter])
apply simp-all
done

```

```

lemma counter-loop-verif:  $(\lambda s. s \downarrow_V''counter''=0 \ \wedge \ s \downarrow_V''t''=0 \ \wedge \ s \downarrow_V''time''=0) \leq |$ 
  LOOP
    (IF  $(\lambda s. s \downarrow_V''t'' = 0.01)$  THEN
       $(\downarrow_V''time'' ::= (\lambda s. s \downarrow_V''time'' + s \downarrow_V''t''))$ ;
       $(\downarrow_V''t'' ::= (\lambda s. 0))$ ;
       $(\downarrow_V''counter'' ::= (\lambda s. s \downarrow_V''counter'' + 1))$ 
    ELSE
       $(x' = f \ \& \ (\lambda s. s \downarrow_V''t'' \leq 0.01)))$ 
    INV  $(\lambda s. s \downarrow_V''time'' = 0.01 * s \downarrow_V''counter'' \ \wedge \ s \downarrow_V''t'' \leq 0.01)$ ]
 $(\lambda s. s \downarrow_V''time'' = 0.01 * s \downarrow_V''counter'')$ 
apply(rule fbox-loopI)
  apply(clarsimp simp: to-var-inject)
  apply(clarsimp simp: to-var-inject)
apply(simp only: fbox-if-then-else, safe)
  apply(clarsimp simp: to-var-inject)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-time-counter])
apply(auto simp: to-var-inject)
done

```

```

no-notation to-var  $(\downarrow_V)$ 
  and val-time-counter (infixl  $\downarrow_V$  90)
  and time-counter-vec-field (f)
  and time-counter-flow ( $\varphi$ )

```

### 0.17.1 PI controller on kinematics

```

abbreviation kin-PI-strs  $\equiv \{''t'', ''r'', ''v'', ''T'', ''roll'', ''error'',$ 
   $''roll-rate'', ''error-sum'', ''veri-test'', ''counter''\}$ 

```

```

typedef kin-PI-vars = kin-PI-strs
  morphisms to-str to-var
  by blast

```

```

notation to-var  $(\downarrow_V)$ 

```

**abbreviation** *kin-PI-terms*  $\equiv \{\downarrow_V''t'', \downarrow_V''r'', \downarrow_V''v'', \downarrow_V''T'', \downarrow_V''roll'', \downarrow_V''error'', \downarrow_V''roll-rate'', \downarrow_V''error-sum'', \downarrow_V''veri-test'', \downarrow_V''counter''\}$

**lemma** *number-of-program-vars*:  $CARD(kin-PI-vars) = 10$   
**using** *type-definition.card type-definition-kin-PI-vars* **by** *fastforce*

**instance** *kin-PI-vars::finite*  
**apply**(*standard, subst bij-betw-finite*[*of to-str UNIV kin-PI-strings*])  
**apply**(*rule bij-betwI'*)  
**apply** (*simp add: to-str-inject*)  
**using** *to-str* **apply** *blast*  
**apply** (*metis to-var-inverse UNIV-I*)  
**by** *simp*

**abbreviation** *val-kin-PI*  $:: real^kin-PI-vars \Rightarrow string \Rightarrow real$  (**infixl**  $\downarrow_V$  90)  
**where**  $s \downarrow_V string \equiv s \$ (to-var string)$

**lemma** *kin-PI-vars-univ*:  $(UNIV::kin-PI-vars\ set) = kin-PI-terms$   
**by** *auto* (*metis kin-PI-vars.to-str kin-PI-vars.to-str-inverse insertE singletonD*)

**lemma** *kin-PI-vars-exhaust*:  $x = \downarrow_V''t'' \vee x = \downarrow_V''r'' \vee x = \downarrow_V''v'' \vee x = \downarrow_V''T''$   
 $\vee x = \downarrow_V''roll'' \vee x = \downarrow_V''roll-rate'' \vee x = \downarrow_V''error'' \vee x = \downarrow_V''error-sum''$   
 $\vee x = \downarrow_V''veri-test'' \vee x = \downarrow_V''counter''$   
**using** *kin-PI-vars-univ* **by** *auto*

**lemma** *kin-PI-vars-sum*:  
**fixes**  $f :: kin-PI-vars \Rightarrow ('a::banach)$   
**shows**  $(\sum_{i \in UNIV}. f\ i) = f(\downarrow_V''t'') + f(\downarrow_V''r'') + f(\downarrow_V''v'') + f(\downarrow_V''T'')$   
 $+ f(\downarrow_V''roll'') + f(\downarrow_V''roll-rate'') + f(\downarrow_V''error'') + f(\downarrow_V''error-sum'')$   
 $+ f(\downarrow_V''veri-test'') + f(\downarrow_V''counter'')$   
**unfolding** *kin-PI-vars-univ* **by** (*simp add: to-var-inject*)

**lemma** *kin-PI-vars-induct*:  $P(\downarrow_V''t'') \Rightarrow P(\downarrow_V''r'') \Rightarrow P(\downarrow_V''v'') \Rightarrow P(\downarrow_V''T'')$   
 $\Rightarrow P(\downarrow_V''roll'') \Rightarrow P(\downarrow_V''roll-rate'') \Rightarrow P(\downarrow_V''error'') \Rightarrow P(\downarrow_V''error-sum'')$   
 $\Rightarrow P(\downarrow_V''veri-test'') \Rightarrow P(\downarrow_V''counter'') \Rightarrow \forall i. P\ i$   
**using** *kin-PI-vars-exhaust* **by** *metis*

**lemma** *kin-PI-vars-eq*:  
 $(s1 = s2) \iff (s1 \downarrow_V''t'' = s2 \downarrow_V''t'' \wedge s1 \downarrow_V''r'' = s2 \downarrow_V''r'' \wedge s1 \downarrow_V''v'' = s2 \downarrow_V''v'' \wedge$   
 $s1 \downarrow_V''T'' = s2 \downarrow_V''T'' \wedge s1 \downarrow_V''roll'' = s2 \downarrow_V''roll'' \wedge s1 \downarrow_V''roll-rate'' = s2 \downarrow_V''roll-rate'' \wedge$   
 $s1 \downarrow_V''error'' = s2 \downarrow_V''error'' \wedge s1 \downarrow_V''error-sum'' = s2 \downarrow_V''error-sum'' \wedge$   
 $s1 \downarrow_V''veri-test'' = s2 \downarrow_V''veri-test'' \wedge s1 \downarrow_V''counter'' = s2 \downarrow_V''counter'')$   
**apply**(*clarsimp simp: vec-eq-iff, rule iffI, force*)  
**by** (*rule kin-PI-vars-induct, auto*)

**abbreviation** *kin-PI-vec-field*  $:: real^kin-PI-vars \Rightarrow real^kin-PI-vars\ (f)$   
**where**  $f\ s \equiv (\chi\ i. \text{if } i = \downarrow_V''t'' \text{ then } 1 \text{ else}$   
 $(\text{if } i = \downarrow_V''r'' \text{ then } s \downarrow_V''v'' \text{ else}$   
 $(\text{if } i = \downarrow_V''v'' \text{ then } s \downarrow_V''T'' \text{ else } 0)))$

**abbreviation** *kin-PI-flow*  $:: real \Rightarrow real^kin-PI-vars \Rightarrow real^kin-PI-vars\ (\varphi)$   
**where**  $\varphi\ t\ s \equiv$   
 $(\chi\ i. \text{if } i = \downarrow_V''t'' \text{ then } t + s \downarrow_V''t'' \text{ else}$   
 $(\text{if } i = \downarrow_V''r'' \text{ then } (s \downarrow_V''T'') * t^2/2 + (s \downarrow_V''v'') * t + (s \downarrow_V''r'') \text{ else}$   
 $(\text{if } i = \downarrow_V''v'' \text{ then } (s \downarrow_V''T'') * t + (s \downarrow_V''v'') \text{ else } s \$ i)))$

**lemma** *local-lipschitz-kin-PI-vec-field*: *local-lipschitz UNIV UNIV* ( $\lambda t. f$ )  
**apply**(*unfold local-lipschitz-def lipschitz-on-def dist-norm*)  
**apply**(*clarsimp, rule-tac x=1 in exI, clarsimp, rule-tac x=1 in exI*)  
**apply**(*clarsimp simp: norm-vec-def L2-set-def*)

**unfolding** *kin-PI-vars-sum* **by** (*simp add: to-var-inject*)

**lemma** *local-flow-kin-PI*: *local-flow f UNIV UNIV  $\varphi$*   
**apply**(*unfold-locales, simp-all add: vec-eq-iff local-lipschitz-kin-PI-vec-field*)  
**apply**(*rule kin-PI-vars-induct, simp-all add: to-var-inject*)  
**by** (*auto intro!: poly-derivatives*)

**lemma** *the-inv-real-Suc* [*simp*]: *the-inv real (real m + 1) = Suc m*  
**by** (*metis Groups.add-ac(2) Num.of-nat-simps(3) inj-def of-nat-eq-iff the-inv-f-f*)

**lemma** *real  $\circ$  the-inv real = id*  
**oops**

**lemma** *the-inv real m = 0  $\implies$  m  $\in$   $\mathbb{N}$*   
**proof**(*subgoal-tac m = 0, simp*)  
**assume** *h0: the-inv real m = 0*  
**have** *0 = real 0*  
**by** *simp*  
**also have** ... = *real (the-inv real m)*  
**using** *h0 arg-cong[of - - real]* **by** *simp*  
**also have** ... = *m*  
**oops**

**lemma** *inNatsD*:  
**assumes** (*n::real*)  $\in \mathbb{N}$   
**shows** *n = 0  $\vee$  n  $\geq$  1*  
**and**  $\exists k::nat. the-inv\ of-nat\ n = k \wedge ((k = 0) \vee (\exists m. k = Suc\ m))$   
**proof**–  
**obtain** *n'::nat* **where** *obs: n = real n'*  
**using** *assms unfolding Nats-def* **by** *blast*  
**have** *n' = 0  $\vee$  n'  $\geq$  1*  
**by** *auto*  
**thus** *n = 0  $\vee$  n  $\geq$  1*  
**using** *obs* **by** (*simp add: real-of-nat-ge-one-iff*)  
**then show**  $\exists k::nat. the-inv\ of-nat\ n = k \wedge ((k = 0) \vee (\exists m. k = Suc\ m))$   
**using** *not0-implies-Suc* **by** *fastforce*  
**qed**

**lemma** *PI-controller-invariants*:

**shows** ( $\lambda s. s \downarrow_V''t''=0 \wedge s \downarrow_V''r''=0 \wedge s \downarrow_V''v''=0 \wedge s \downarrow_V''T''=1 \wedge s \downarrow_V''roll''=0 \wedge s \downarrow_V''roll-rate''=0 \wedge$   
 $s \downarrow_V''error''=0 \wedge s \downarrow_V''error-sum''=0 \wedge s \downarrow_V''veri-test''=0 \wedge s \downarrow_V''counter''=0$ )  $\leq$   
 $|LOOP$   
 (*IF* ( $\lambda s. s \downarrow_V''t'' = dt$ ) *THEN*  
 — **CONTROL**  
 ( $\downarrow_V''error'' ::= (\lambda s. s \downarrow_V''r'' - s \downarrow_V''roll'')$ );  
 ( $\downarrow_V''error-sum'' ::= (\lambda s. s \downarrow_V''error-sum'' + s \downarrow_V''error'')$ );  
 ( $\downarrow_V''T'' ::= (\lambda s. Prop * s \downarrow_V''error'' + Integr * dt * s \downarrow_V''error-sum'')$ );  
 ( $\downarrow_V''roll'' ::= (\lambda s. s \downarrow_V''r'')$ );  
 ( $\downarrow_V''roll-rate'' ::= (\lambda s. s \downarrow_V''v'')$ );  
 ( $\downarrow_V''veri-test'' ::= (\lambda s. s \downarrow_V''veri-test'' + s \downarrow_V''error-sum'')$ );  
 ( $\downarrow_V''counter'' ::= (\lambda s. s \downarrow_V''counter'' + 1)$ );  
 ( $\downarrow_V''t'' ::= (\lambda s. 0)$ )  
*ELSE*  
 — **DYNAMICS**  
 ( $x' = f \ \& \ (\lambda s. s \downarrow_V''t'' \leq dt)$ ))  
*INV* ( $\lambda s. (s \downarrow_V''counter'' \geq 1 \longrightarrow s \downarrow_V''T'' = Prop * s \downarrow_V''error'' + Integr * dt * s \downarrow_V''error-sum'') \wedge$   
 $s \downarrow_V''counter'' \in \mathbb{N} \wedge s \downarrow_V''v'' = s \downarrow_V''T'' * s \downarrow_V''t'' + s \downarrow_V''roll-rate'' \wedge$   
 $s \downarrow_V''r'' = s \downarrow_V''T'' * s \downarrow_V''t''^2/2 + s \downarrow_V''roll-rate'' * s \downarrow_V''t'' + s \downarrow_V''roll'')$ )  
 $(\lambda s. (s \downarrow_V''counter'' \geq 1 \longrightarrow s \downarrow_V''T'' = Prop * s \downarrow_V''error'' + Integr * dt * s \downarrow_V''error-sum'') \wedge$

```

 $s|_V''counter'' \in \mathbb{N} \wedge s|_V''v'' = s|_V''T'' * s|_V''t'' + s|_V''roll-rate'' \wedge$ 
 $s|_V''r'' = s|_V''T'' * s|_V''t''^2/2 + s|_V''roll-rate'' * s|_V''t'' + s|_V''roll''$ 
apply(rule fbox-loopI)
  apply (clarsimp simp: to-var-inject)
  apply (clarsimp simp: to-var-inject)
apply(subst fbox-if-then-else, clarify, rule conjI)
  apply (clarsimp simp: to-var-inject power2-sum)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-kin-PI], simp)
apply(clarsimp simp: le-fun-def to-var-inject power2-sum)
  apply(force simp: field-simps)
done

lemma PI-controller-invariants2:
  assumes Prop = -6 and Integr = -0.1 and dt = 0.01
  shows ( $\lambda s. s|_V''t''=0 \wedge s|_V''v'' = s|_V''T'' * s|_V''t'' + s|_V''roll-rate'' \wedge s|_V''counter'' \in \mathbb{N} \wedge$ 
 $s|_V''r'' = s|_V''T'' * s|_V''t''^2/2 + s|_V''roll-rate'' * s|_V''t'' + s|_V''roll'' \wedge$ 
 $s|_V''T'' = Prop * s|_V''error'' + Integr * dt * s|_V''error-sum'' \wedge s|_V''error-sum'' - s|_V''error'' \leq 2/10$ 
 $\wedge$ 
 $s|_V''roll-rate'' = Prop * dt * (s|_V''error-sum'' - s|_V''error'') + Integr * dt^2 * (s|_V''veri-test'' -$ 
 $s|_V''error-sum'')$ )  $\leq$ 
  |LOOP
    (IF ( $\lambda s. s|_V''t'' = dt$ ) THEN
      — CONTROL
      ( $|_V''error'' ::= (\lambda s. s|_V''r'' - s|_V''roll'')$ );
      ( $|_V''error-sum'' ::= (\lambda s. s|_V''error-sum'' + s|_V''error'')$ );
      ( $|_V''T'' ::= (\lambda s. Prop * s|_V''error'' + Integr * dt * s|_V''error-sum'')$ );
      ( $|_V''roll'' ::= (\lambda s. s|_V''r'')$ );
      ( $|_V''roll-rate'' ::= (\lambda s. s|_V''v'')$ );
      ( $|_V''veri-test'' ::= (\lambda s. s|_V''veri-test'' + s|_V''error-sum'')$ );
      ( $|_V''t'' ::= (\lambda s. 0)$ )
    ELSE
      — DYNAMICS
      ( $x' = f \ \& \ (\lambda s. 0 \leq s|_V''t'' \wedge s|_V''t'' \leq dt)$ ))
  INV ( $\lambda s. s|_V''T'' = Prop * s|_V''error'' + Integr * dt * s|_V''error-sum'' \wedge$ 
 $s|_V''roll-rate'' = Prop * dt * (s|_V''error-sum'' - s|_V''error'') + Integr * dt^2 * (s|_V''veri-test'' -$ 
 $s|_V''error-sum'')$ )  $\wedge$ 
 $s|_V''counter'' \in \mathbb{N} \wedge s|_V''v'' = s|_V''T'' * s|_V''t'' + s|_V''roll-rate'' \wedge$ 
 $s|_V''r'' = s|_V''T'' * s|_V''t''^2/2 + s|_V''roll-rate'' * s|_V''t'' + s|_V''roll''$ )
  ( $\lambda s. s|_V''T'' = Prop * s|_V''error'' + Integr * dt * s|_V''error-sum'' \wedge$ 
 $s|_V''counter'' \in \mathbb{N} \wedge s|_V''v'' = s|_V''T'' * s|_V''t'' + s|_V''roll-rate'' \wedge$ 
 $s|_V''r'' = s|_V''T'' * s|_V''t''^2/2 + s|_V''roll-rate'' * s|_V''t'' + s|_V''roll''$ )
apply(rule fbox-loopI)
  apply (clarsimp simp: to-var-inject)
  apply (clarsimp simp: to-var-inject)
apply(subst fbox-if-then-else, clarify, rule conjI)
  apply (force simp: to-var-inject power2-eq-square field-simps)
apply(subst local-flow.fbox-g-ode-subset[OF local-flow-kin-PI], simp)
apply(clarsimp simp: le-fun-def to-var-inject power2-sum)
  apply(force simp: field-simps)
done

end

```

## 0.18 Verification components with Kleene Algebras

We create verification rules based on various Kleene Algebras.

```

theory VC-diffKAD-KA
imports
  KAT-and-DRA.PHL-KAT
  KAD.Modal-Kleene-Algebra
  Transformer-Semantics.Kleisli-Quantale

```

```

begin

```

### 0.18.1 Hoare logic and refinement in KAT

Here we derive the rules of Hoare Logic and a refinement calculus in Kleene algebra with tests.

```

notation  $t$  ( $\mathsf{tt}$ )

```

```

hide-const  $t$ 

```

```

no-notation  $\mathsf{ars-r}$  ( $r$ )
  and  $\mathsf{if-then-else}$  ( $\mathsf{if} - \mathsf{then} - \mathsf{else} - \mathsf{fi}$  [ $64, 64, 64$ ]  $63$ )
  and  $\mathsf{while}$  ( $\mathsf{while} - \mathsf{do} - \mathsf{od}$  [ $64, 64$ ]  $63$ )

```

```

context  $\mathsf{kat}$ 
begin

```

— Definitions of Hoare Triple

```

definition  $\mathsf{Hoare} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \mathsf{bool}$  ( $H$ ) where
   $H\ p\ x\ q \longleftrightarrow \mathsf{tt}\ p \cdot x \leq x \cdot \mathsf{tt}\ q$ 

```

```

lemma  $\mathsf{H-consl}$ :  $\mathsf{tt}\ p \leq \mathsf{tt}\ p' \Longrightarrow H\ p'\ x\ q \Longrightarrow H\ p\ x\ q$ 
using  $\mathsf{Hoare-def}\ \mathsf{phl-cons1}$  by  $\mathsf{blast}$ 

```

```

lemma  $\mathsf{H-consr}$ :  $\mathsf{tt}\ q' \leq \mathsf{tt}\ q \Longrightarrow H\ p\ x\ q' \Longrightarrow H\ p\ x\ q$ 
using  $\mathsf{Hoare-def}\ \mathsf{phl-cons2}$  by  $\mathsf{blast}$ 

```

```

lemma  $\mathsf{H-cons}$ :  $\mathsf{tt}\ p \leq \mathsf{tt}\ p' \Longrightarrow \mathsf{tt}\ q' \leq \mathsf{tt}\ q \Longrightarrow H\ p'\ x\ q' \Longrightarrow H\ p\ x\ q$ 
by ( $\mathsf{simp}\ \mathsf{add}$ :  $\mathsf{H-consl}\ \mathsf{H-consr}$ )

```

— Skip program

```

lemma  $\mathsf{H-skip}$ :  $H\ p\ 1\ p$ 
by ( $\mathsf{simp}\ \mathsf{add}$ :  $\mathsf{Hoare-def}$ )

```

— Sequential composition

```

lemma  $\mathsf{H-seq}$ :  $H\ p\ x\ r \Longrightarrow H\ r\ y\ q \Longrightarrow H\ p\ (x \cdot y)\ q$ 
by ( $\mathsf{simp}\ \mathsf{add}$ :  $\mathsf{Hoare-def}\ \mathsf{phl-seq}$ )

```

— Conditional statement

```

definition  $\mathsf{kat-cond} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$  ( $\mathsf{if} - \mathsf{then} - \mathsf{else} - \mathsf{fi}$  [ $64, 64, 64$ ]  $63$ ) where
   $\mathsf{if}\ p\ \mathsf{then}\ x\ \mathsf{else}\ y\ \mathsf{fi} = (\mathsf{tt}\ p \cdot x + \mathsf{tt}\ p \cdot y)$ 

```

```

lemma  $\mathsf{H-var}$ :  $H\ p\ x\ q \longleftrightarrow \mathsf{tt}\ p \cdot x \cdot n\ q = 0$ 
by ( $\mathsf{metis}\ \mathsf{Hoare-def}\ \mathsf{n-kat-3}\ \mathsf{t-n-closed}$ )

```

```

lemma  $\mathsf{H-cond-iff}$ :  $H\ p\ (\mathsf{if}\ r\ \mathsf{then}\ x\ \mathsf{else}\ y\ \mathsf{fi})\ q \longleftrightarrow H\ (\mathsf{tt}\ p \cdot \mathsf{tt}\ r)\ x\ q \wedge H\ (\mathsf{tt}\ p \cdot \mathsf{tt}\ r)\ y\ q$ 
proof –

```

```

  have  $H\ p\ (\mathsf{if}\ r\ \mathsf{then}\ x\ \mathsf{else}\ y\ \mathsf{fi})\ q \longleftrightarrow \mathsf{tt}\ p \cdot (\mathsf{tt}\ r \cdot x + \mathsf{tt}\ r \cdot y) \cdot n\ q = 0$ 
  by ( $\mathsf{simp}\ \mathsf{add}$ :  $\mathsf{H-var}\ \mathsf{kat-cond-def}$ )

```

```

  also have  $\dots \longleftrightarrow \mathsf{tt}\ p \cdot \mathsf{tt}\ r \cdot x \cdot n\ q + \mathsf{tt}\ p \cdot \mathsf{tt}\ r \cdot y \cdot n\ q = 0$ 

```

by (simp add: distrib-left mult-assoc)  
 also have ...  $\longleftrightarrow \text{tt } p \cdot \text{tt } r \cdot x \cdot n \ q = 0 \wedge \text{tt } p \cdot n \ r \cdot y \cdot n \ q = 0$   
 by (metis add-0-left no-trivial-inverse)  
 finally show ?thesis  
 by (metis H-var test-mult)  
 qed

lemma H-cond:  $H (\text{tt } p \cdot \text{tt } r) \ x \ q \implies H (\text{tt } p \cdot n \ r) \ y \ q \implies H \ p \ (\text{if } r \text{ then } x \text{ else } y \ \text{fi}) \ q$   
 by (simp add: H-cond-iff)

— While loop

definition kat-while ::  $'a \Rightarrow 'a \Rightarrow 'a \ (while - do - od \ [64,64] \ 63)$  where  
 $while \ b \ do \ x \ od = (\text{tt } b \cdot x)^* \cdot n \ b$

definition kat-while-inv ::  $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \ (while - inv - do - od \ [64,64,64] \ 63)$  where  
 $while \ p \ inv \ i \ do \ x \ od = while \ p \ do \ x \ od$

lemma H-exp1:  $H (\text{tt } p \cdot \text{tt } r) \ x \ q \implies H \ p \ (\text{tt } r \cdot x) \ q$   
 using Hoare-def n-de-morgan-var2 phl.ht-at-phl-export1 by auto

lemma H-while:  $H (\text{tt } p \cdot \text{tt } r) \ x \ p \implies H \ p \ (while \ r \ do \ x \ od) (\text{tt } p \cdot n \ r)$

proof –

assume a1:  $H (\text{tt } p \cdot \text{tt } r) \ x \ p$   
 have  $\text{tt } (\text{tt } p \cdot n \ r) = n \ r \cdot \text{tt } p \cdot n \ r$   
 using n-preserve test-mult by presburger  
 then show ?thesis  
 using a1 Hoare-def H-exp1 conway.phl.it-simr phl-export2 kat-while-def by auto

qed

lemma H-while-inv:  $\text{tt } p \leq \text{tt } i \implies \text{tt } i \cdot n \ r \leq \text{tt } q \implies H (\text{tt } i \cdot \text{tt } r) \ x \ i \implies H \ p \ (while \ r \ inv \ i \ do \ x \ od) \ q$   
 by (metis H-cons H-while test-mult kat-while-inv-def)

— Finite iteration

lemma H-star:  $H \ i \ x \ i \implies H \ i \ (x^*) \ i$   
 unfolding Hoare-def using star-sim2 by blast

lemma H-star-inv:  
 assumes  $\text{tt } p \leq \text{tt } i$  and  $H \ i \ x \ i$  and  $(\text{tt } i) \leq (\text{tt } q)$   
 shows  $H \ p \ (x^*) \ q$

proof –

have  $H \ i \ (x^*) \ i$   
 using assms(2) H-star by blast  
 hence  $H \ p \ (x^*) \ i$   
 unfolding Hoare-def using assms(1) phl-cons1 by blast  
 thus ?thesis  
 unfolding Hoare-def using assms(3) phl-cons2 by blast

qed

definition kat-loop-inv ::  $'a \Rightarrow 'a \Rightarrow 'a \ (loop - inv - [64,64] \ 63)$   
 where  $loop \ x \ inv \ i = x^*$

lemma H-loop:  $H \ p \ x \ p \implies H \ p \ (loop \ x \ inv \ i) \ p$   
 unfolding kat-loop-inv-def by (rule H-star)

lemma H-loop-inv:  $\text{tt } p \leq \text{tt } i \implies H \ i \ x \ i \implies \text{tt } i \leq \text{tt } q \implies H \ p \ (loop \ x \ inv \ i) \ q$   
 unfolding kat-loop-inv-def using H-star-inv by blast

— Invariants



**lemma** *H-inv*:  $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies H \ i \ x \ i \implies H \ p \ x \ q$   
**by** (*rule-tac* *p'=i* **and** *q'=i* **in** *H-cons*)

**lemma** *H-inv-plus*:  $\text{tt } i = i \implies \text{tt } j = j \implies H \ i \ x \ i \implies H \ j \ x \ j \implies H \ (i + j) \ x \ (i + j)$   
**unfolding** *Hoare-def* **using** *combine-common-factor*  
**by** (*smt add-commute add.left-commute distrib-left join.sup.absorb-iff1 t-add-closed*)

**lemma** *H-inv-mult*:  $\text{tt } i = i \implies \text{tt } j = j \implies H \ i \ x \ i \implies H \ j \ x \ j \implies H \ (i \cdot j) \ x \ (i \cdot j)$   
**unfolding** *Hoare-def* **by** (*smt n-kat-2 n-mult-comm t-mult-closure mult-assoc*)

**end**

### 0.18.2 refinement KAT

**class** *rkate* = *kate* +  
**fixes** *Ref* ::  $'a \Rightarrow 'a \Rightarrow 'a$   
**assumes** *spec-def*:  $x \leq \text{Ref } p \ q \longleftrightarrow H \ p \ x \ q$

**begin**

**lemma** *R1*:  $H \ p \ (\text{Ref } p \ q) \ q$   
**using** *spec-def* **by** *blast*

**lemma** *R2*:  $H \ p \ x \ q \implies x \leq \text{Ref } p \ q$   
**by** (*simp add: spec-def*)

**lemma** *R-cons*:  $\text{tt } p \leq \text{tt } p' \implies \text{tt } q' \leq \text{tt } q \implies \text{Ref } p' \ q' \leq \text{Ref } p \ q$

**proof** —

**assume** *h1*:  $\text{tt } p \leq \text{tt } p'$  **and** *h2*:  $\text{tt } q' \leq \text{tt } q$

**have**  $H \ p' \ (\text{Ref } p' \ q') \ q'$

**by** (*simp add: R1*)

**hence**  $H \ p \ (\text{Ref } p' \ q') \ q$

**using** *h1 h2 H-consl H-consr* **by** *blast*

**thus** *?thesis*

**by** (*rule R2*)

**qed**

— Abort and skip programs

**lemma** *R-skip*:  $1 \leq \text{Ref } p \ p$

**proof** —

**have**  $H \ p \ 1 \ p$

**by** (*simp add: H-skip*)

**thus** *?thesis*

**by** (*rule R2*)

**qed**

**lemma** *R-zero-one*:  $x \leq \text{Ref } 0 \ 1$

**proof** —

**have**  $H \ 0 \ x \ 1$

**by** (*simp add: Hoare-def*)

**thus** *?thesis*

**by** (*rule R2*)

**qed**

**lemma** *R-one-zero*:  $\text{Ref } 1 \ 0 = 0$

**proof** —

**have**  $H \ 1 \ (\text{Ref } 1 \ 0) \ 0$

**by** (*simp add: R1*)

**thus** *?thesis*  
**by** (*simp add: Hoare-def join.le-bot*)  
**qed**

— Sequential composition

**lemma** *R-seq*:  $(\text{Ref } p \ r) \cdot (\text{Ref } r \ q) \leq \text{Ref } p \ q$

**proof** —

**have**  $H \ p \ (\text{Ref } p \ r) \ r$  **and**  $H \ r \ (\text{Ref } r \ q) \ q$   
**by** (*simp add: R1*)  
**hence**  $H \ p \ ((\text{Ref } p \ r) \cdot (\text{Ref } r \ q)) \ q$   
**by** (*rule H-seq*)  
**thus** *?thesis*  
**by** (*rule R2*)  
**qed**

— Conditional statement

**lemma** *R-cond*: *if*  $v$  *then*  $(\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q)$  *else*  $(\text{Ref } (n \ v \cdot \text{tt } p) \ q)$  *fi*  $\leq \text{Ref } p \ q$

**proof** —

**have**  $H \ (\text{tt } v \cdot \text{tt } p) \ (\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q) \ q$  **and**  $H \ (n \ v \cdot \text{tt } p) \ (\text{Ref } (n \ v \cdot \text{tt } p) \ q) \ q$   
**by** (*simp add: R1*)  
**hence**  $H \ p \ (\text{if } v \text{ then } (\text{Ref } (\text{tt } v \cdot \text{tt } p) \ q) \text{ else } (\text{Ref } (n \ v \cdot \text{tt } p) \ q) \text{ fi}) \ q$   
**by** (*simp add: H-cond n-mult-comm*)  
**thus** *?thesis*  
**by** (*rule R2*)  
**qed**

— While loop

**lemma** *R-while*: *while*  $q$  *do*  $(\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p)$  *od*  $\leq \text{Ref } p \ (\text{tt } p \cdot n \ q)$

**proof** —

**have**  $H \ (\text{tt } p \cdot \text{tt } q) \ (\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p) \ p$   
**by** (*simp-all add: R1*)  
**hence**  $H \ p \ (\text{while } q \text{ do } (\text{Ref } (\text{tt } p \cdot \text{tt } q) \ p) \text{ od}) \ (\text{tt } p \cdot n \ q)$   
**by** (*simp add: H-while*)  
**thus** *?thesis*  
**by** (*rule R2*)  
**qed**

— Finite iteration

**lemma** *R-star*:  $(\text{Ref } i \ i)^* \leq \text{Ref } i \ i$

**proof** —

**have**  $H \ i \ (\text{Ref } i \ i) \ i$   
**using** *R1* **by** *blast*  
**hence**  $H \ i \ ((\text{Ref } i \ i)^*) \ i$   
**using** *H-star* **by** *blast*  
**thus**  $\text{Ref } i \ i^* \leq \text{Ref } i \ i$   
**by** (*rule R2*)  
**qed**

**lemma** *R-loop*: *loop*  $(\text{Ref } p \ p)$  *inv*  $i \leq \text{Ref } p \ p$

**unfolding** *kat-loop-inv-def* **by** (*rule R-star*)

— Invariants

**lemma** *R-inv*:  $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies \text{Ref } i \ i \leq \text{Ref } p \ q$

**using** *R-cons* **by** *force*

end

**no-notation** *kat-cond* (*if - then - else - fi* [64,64,64] 63)  
**and** *kat-while* (*while - do - od* [64,64] 63)  
**and** *kat-while-inv* (*while - inv - do - od* [64,64,64] 63)  
**and** *kat-loop-inv* (*loop - inv -* [64,64] 63)

### 0.18.3 Verification in AKA (KAD)

Here we derive verification components with weakest liberal preconditions based on antidomain Kleene algebra (or Kleene algebra with domain)

**context** *antidomain-kleene-algebra*  
**begin**

— Sequential composition

**declare** *fbox-mult* [*simp*]

— Conditional statement

**definition** *aka-cond* :: '*a*  $\Rightarrow$  '*a*  $\Rightarrow$  '*a*  $\Rightarrow$  '*a* (*if - then - else - fi* [64,64,64] 63)  
**where** *if p then x else y fi* = *d p* · *x* + *ad p* · *y*

**lemma** *fbox-export1*: *ad p* + [*x*] *q* = [*d p* · *x*] *q*  
**using** *a-d-add-closure addual.ars-r-def fbox-def fbox-mult* **by** *auto*

**lemma** *fbox-cond* [*simp*]: [*if p then x else y fi*] *q* = (*ad p* + [*x*] *q*) · (*d p* + [*y*] *q*)  
**using** *aka-cond-def a-closure' ads-d-def ans-d-def fbox-add2 fbox-export1* **by** *auto*

— Finite iteration

**definition** *aka-loop-inv* :: '*a*  $\Rightarrow$  '*a*  $\Rightarrow$  '*a* (*loop - inv -* [64,64] 63)  
**where** *loop x inv i* = *x*<sup>\*</sup>

**lemma** *fbox-stari*: *d p* ≤ *d i*  $\implies$  *d i* ≤ [*x*] *i*  $\implies$  *d i* ≤ *d q*  $\implies$  *d p* ≤ [*x*<sup>\*</sup>] *q*  
**by** (*meson dual-order.trans fbox-iso fbox-star-induct-var*)

**lemma** *fbox-loopi*: *d p* ≤ *d i*  $\implies$  *d i* ≤ [*x*] *i*  $\implies$  *d i* ≤ *d q*  $\implies$  *d p* ≤ [*loop x inv i*] *q*  
**unfolding** *aka-loop-inv-def* **using** *fbox-stari* **by** *blast*

— Invariants

**lemma** *fbox-frame*: *d p* · *x* ≤ *x* · *d p*  $\implies$  *d q* ≤ [*x*] *r*  $\implies$  *d p* · *d q* ≤ [*x*] (*d p* · *d r*)  
**using** *dual.mult-isol-var fbox-add1 fbox-demodalisation3 fbox-simp* **by** *auto*

**lemma** *plus-inv*: *i* ≤ [*x*] *i*  $\implies$  *j* ≤ [*x*] *j*  $\implies$  (*i* + *j*) ≤ [*x*] (*i* + *j*)  
**by** (*metis ads-d-def dka.dsr5 fbox-simp fbox-subdist join.sup-mono order-trans*)

**lemma** *mult-inv*: *d i* ≤ [*x*] *d i*  $\implies$  *d j* ≤ [*x*] *d j*  $\implies$  (*d i* · *d j*) ≤ [*x*] (*d i* · *d j*)  
**using** *fbox-demodalisation3 fbox-frame fbox-simp* **by** *auto*

end

### 0.18.4 Relational model

We show that relations form Kleene Algebras (KAT and AKA).

**interpretation** *rel-ug*: *unital-quantale Id (O)  $\cap$   $\cup$  ( $\cap$ ) ( $\subseteq$ ) ( $\subset$ ) ( $\cup$ )  $\{\}$  UNIV*  
**by** (*unfold-locales, auto*)

**lemma** *power-is-relpow*:  $\text{rel-}uq.\text{power } X \ m = X^{\wedge m}$  **for**  $X :: 'a \text{ rel}$

**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*:  $X^* = (\bigcup m. \text{rel-}uq.\text{power } X \ m)$

**by** (*simp add: power-is-relpow rtrancl-is-UN-relpow*)

**lemma** *rel-star-contl*:  $X \ O \ Y^* = (\bigcup m. X \ O \ \text{rel-}uq.\text{power } Y \ m)$

**by** (*metis rel-star-def relcomp-UNION-distrib*)

**lemma** *rel-star-contr*:  $X^* \ O \ Y = (\bigcup m. (\text{rel-}uq.\text{power } X \ m) \ O \ Y)$

**by** (*metis rel-star-def relcomp-UNION-distrib2*)

**interpretation** *rel-ka*: *kleene-algebra*  $(\cup) \ (O) \ Id \ \{\} \ (\subseteq) \ (\subset) \ rtrancl$

**proof**

**fix**  $x \ y \ z :: 'a \text{ rel}$

**show**  $Id \cup x \ O \ x^* \subseteq x^*$

**by** (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)

**next**

**fix**  $x \ y \ z :: 'a \text{ rel}$

**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**  $x \ y \ z :: 'a \text{ rel}$

**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* ::  $'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel}$  **where**

$\text{rel-}R \ P \ Q = \bigcup \{X. \text{rel-}kat.Hoare \ P \ X \ Q\}$

**interpretation** *rel-rkat*: *rkat*  $(\cup) \ (;) \ Id \ \{\} \ (\subseteq) \ (\subset) \ rtrancl \ (\lambda X. Id \cap - \ X) \ \text{rel-}R$

**by** (*standard, auto simp: rel-R-def rel-kat.Hoare-def*)

**lemma** *RdL-is-rRKAT*:  $(\forall x. \{(x, x)\}; R1 \subseteq \{(x, x)\}; R2) = (R1 \subseteq R2)$

**by** *auto*

**definition** *rel-ad* ::  $'a \text{ rel} \Rightarrow 'a \text{ rel}$  **where**

$\text{rel-}ad \ R = \{(x, x) \mid x. \neg (\exists y. (x, y) \in R)\}$

**interpretation** *rel-aka*: *antidomain-kleene-algebra*  $\text{rel-}ad \ (\cup) \ (O) \ Id \ \{\} \ (\subseteq) \ (\subset) \ rtrancl$

**by** *unfold-locales (auto simp: rel-ad-def)*

### 0.18.5 State transformer model

We show that state transformers form Kleene Algebras (KAT and AKA).

**notation** *Abs-nd-fun*  $(-\bullet [101] 100)$   
**and** *Rep-nd-fun*  $(-\bullet [101] 100)$

**declare** *Abs-nd-fun-inverse* [*simp*]

**lemma** *nd-fun-ext*:  $(\bigwedge x. (f\bullet) x = (g\bullet) x) \implies f = g$   
**apply** (*subgoal-tac Rep-nd-fun f = Rep-nd-fun g*)  
**using** *Rep-nd-fun-inject*  
**apply** *blast*  
**by** (*rule ext, simp*)

**lemma** *nd-fun-eq-iff*:  $(f = g) = (\forall x. (f\bullet) x = (g\bullet) x)$   
**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-zero*[*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*]:  $1 + x \cdot x^* \leq x^*$   
**and** *nd-star-unfoldl*[*nd-fun-aka*]:  $z + x \cdot y \leq y \implies x^* \cdot z \leq y$   
**and** *nd-star-unfoldr*[*nd-fun-aka*]:  $z + y \cdot x \leq y \implies z \cdot x^* \leq y$  **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**  $n\ f = (\lambda x. \text{ if } ((f \bullet) x = \{\}) \text{ then } \{x\} \text{ else } \{\})^\bullet$

**lemma**  $nd\text{-fun-}n\text{-op-one}[nd\text{-fun-}aka]: n\ (n\ (1::'a\ nd\text{-fun})) = 1$   
**and**  $nd\text{-fun-}n\text{-op-mult}[nd\text{-fun-}aka]: n\ (n\ (n\ x \cdot n\ y)) = n\ x \cdot n\ y$   
**and**  $nd\text{-fun-}n\text{-op-mult-comp}[nd\text{-fun-}aka]: n\ x \cdot n\ (n\ x) = 0$   
**and**  $nd\text{-fun-}n\text{-op-de-morgan}[nd\text{-fun-}aka]: n\ (n\ (n\ x) \cdot n\ (n\ y)) = n\ x + n\ y$  **for**  $x::'a\ nd\text{-fun}$   
**unfolding**  $n\text{-op-}nd\text{-fun-def}\ one\text{-nd-fun-def}\ times\text{-nd-fun-def}\ plus\text{-nd-fun-def}\ zero\text{-nd-fun-def}$   
**by**  $(\text{auto simp: } nd\text{-fun-eq-iff}\ kcomp\text{-def})$

**instance**

**by**  $(\text{intro-classes, auto simp: } nd\text{-fun-}aka)$

**end**

**instantiation**  $nd\text{-fun} :: (type)\ rkat$

**begin**

**definition**  $Ref\text{-}nd\text{-fun}\ P\ Q \equiv (\lambda s. \bigcup \{(f \bullet) s \mid f. \text{Hoare } P\ f\ Q\})^\bullet$

**instance**

**apply**  $(\text{intro-classes})$

**by**  $(\text{unfold Hoare-def}\ n\text{-op-}nd\text{-fun-def}\ Ref\text{-}nd\text{-fun-def}\ times\text{-nd-fun-def})$

$(\text{auto simp: } kcomp\text{-def}\ le\text{-fun-def}\ less\text{-eq-}nd\text{-fun-def})$

**end**

**instantiation**  $nd\text{-fun} :: (type)\ antidomain\text{-kleene-algebra}$

**begin**

**definition**  $ad\ f = (\lambda x. \text{ if } ((f \bullet) x = \{\}) \text{ then } \{x\} \text{ else } \{\})^\bullet$

**lemma**  $nd\text{-fun-ad-zero}[nd\text{-fun-}aka]: ad\ x \cdot x = 0$   
**and**  $nd\text{-fun-ad}[nd\text{-fun-}aka]: ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$   
**and**  $nd\text{-fun-ad-one}[nd\text{-fun-}aka]: ad\ (ad\ x) + ad\ x = 1$  **for**  $x::'a\ nd\text{-fun}$   
**unfolding**  $antidomain\text{-op-}nd\text{-fun-def}\ times\text{-nd-fun-def}\ plus\text{-nd-fun-def}\ zero\text{-nd-fun-def}$   
**by**  $(\text{auto simp: } nd\text{-fun-eq-iff}\ kcomp\text{-def}\ one\text{-nd-fun-def})$

**instance**

**apply**  $\text{intro-classes}$

**using**  $nd\text{-fun-}aka$  **by**  $\text{simp-all}$

**end**

**end**

## 0.19 VC\_diffKAD

**theory**  $VC\text{-diffKAD-auxiliarities}$

**imports**  $Main\ VC\text{-diffKAD-KA}\ Ordinary\text{-Differential-Equations.ODE-Analysis}$

**begin**

### 0.19.1 Stack Theories Preliminaries: VC\_KAD and ODEs

To make our notation less code-like and more mathematical we declare:

**type-synonym**  $'a\ pred = 'a \Rightarrow bool$

**type-synonym**  $'a\ store = string \Rightarrow 'a$

**hide-const**  $\eta$

**no-notation** *Archimedean-Field.ceiling* ( $\lceil \cdot \rceil$ )  
**and** *Archimedean-Field.floor* ( $\lfloor \cdot \rfloor$ )  
**and** *Set.image* ( $\cdot \mapsto$ )  
**and** *Range-Semiring.antirange-semiring-class.ars-r* ( $r$ )  
**and** *antidomain-semiringl.ads-d* ( $d$ )  
**and** *n-op* ( $n$  -  $[90]$  91)  
**and** *Hoare* ( $H$ )  
**and** *tau* ( $\tau$ )  
**and** *dual* ( $\partial$ )  
**and** *fres* (**infixl**  $\leftarrow$  60)  
**and** *n-add-op* (**infixl**  $\oplus$  65)  
**and** *eta* ( $\eta$ )

**notation** *Set.image* ( $\cdot \mapsto$ )  
**and** *Product-Type.prod.fst* ( $\pi_1$ )  
**and** *Product-Type.prod.snd* ( $\pi_2$ )  
**and** *List.zip* (**infixl**  $\otimes$  63)  
**and** *rel-aka.fbox* ( $wp$ )

**definition**  $p2r :: 'a \text{ pred} \Rightarrow 'a \text{ rel } ((\lceil \cdot \rceil))$  **where**  
 $\lceil P \rceil = \{(s, s) \mid s. P \ s\}$

**lemma**  $p2r\text{-simps}[simp]$ :  
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P \ s \longrightarrow Q \ s)$   
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P \ s = Q \ s)$   
 $(\lceil P \rceil ; \lceil Q \rceil) = \lceil \lambda s. P \ s \wedge Q \ s \rceil$   
 $(\lceil P \rceil \cup \lceil Q \rceil) = \lceil \lambda s. P \ s \vee Q \ s \rceil$   
 $rel\text{-ad } \lceil P \rceil = \lceil \lambda s. \neg P \ s \rceil$   
 $rel\text{-aka.ads-d } \lceil P \rceil = \lceil P \rceil$   
**unfolding**  $p2r\text{-def } rel\text{-ad-def } rel\text{-aka.ads-d-def}$  **by** *auto*

**lemma**  $wp\text{-rel}$ :  $wp \ R \ \lceil P \rceil = \lceil \lambda x. \forall y. (x, y) \in R \longrightarrow P \ y \rceil$   
**unfolding**  $rel\text{-aka.fbox-def } p2r\text{-def } rel\text{-ad-def}$  **by** *auto*

**lemma**  $boxProgrPred\text{-chrctrzn}$ :  $(x, y) \in wp \ R \ \lceil P \rceil \iff (y = x \wedge (\forall y. (x, y) \in R \longrightarrow P \ y))$   
**unfolding**  $wp\text{-rel}$  **unfolding**  $p2r\text{-def}$  **by** *simp*

**definition**  $assign :: string \Rightarrow ('a \text{ store} \Rightarrow 'a) \Rightarrow ('a \text{ store}) \text{ rel } (- ::= -)$   
**where**  $(x ::= e) = \{(s, s(x := e \ s)) \mid s. True\}$

**lemma**  $wp\text{-assign } [simp]$ :  $wp \ (x ::= e) \ \lceil P \rceil = \lceil \lambda s. P \ (s(x := e \ s)) \rceil$   
**unfolding**  $wp\text{-rel } assign\text{-def}$  **by** *simp*

**abbreviation**  $cond\text{-sugar} :: 'a \text{ pred} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow (IF - THEN - ELSE - [64, 64] \ 63)$   
**where**  $IF \ P \ THEN \ X \ ELSE \ Y \equiv rel\text{-aka.aka-cond } \lceil P \rceil \ X \ Y$

**lemma**  $wp\text{-loopI}$ :  $\lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \lceil Q \rceil \implies \lceil I \rceil \leq wp \ R \ \lceil I \rceil \implies \lceil P \rceil \leq wp \ (R^*) \ \lceil Q \rceil$   
**using**  $rel\text{-aka.fbox-stari}[of \ \lceil P \rceil \ \lceil I \rceil]$  **by** *auto*

**proposition**  $cons\text{-eq}\text{-zipE}$ :  
 $(x, y) \# tail = xList \otimes yList \implies \exists xTail \ yTail. x \# xTail = xList \wedge y \# yTail = yList$   
**by**(*induction xList, simp-all, induction yList, simp-all*)

**proposition**  $set\text{-zip-left-rightD}$ :  
 $(x, y) \in set \ (xList \otimes yList) \implies x \in set \ xList \wedge y \in set \ yList$   
**apply**(*rule conjI*)  
**apply**(*rule tac y=y and ys=yList in set-zip-leftD, simp*)

```

apply(rule-tac  $x=x$  and  $xs=xList$  in set-zip-rightD, simp)
done

```

```

declare zip-map-fst-snd [simp]

```

### 0.19.2 VC\_diffKAD Preliminaries

In dL, the set of possible program variables is split in two, the set of variables  $V$  and their primed counterparts  $V'$ . To implement this, we use Isabelle's string-type and define a function that primes a given string. We then define the set of primed-strings based on it.

```

definition vdiff :: string  $\Rightarrow$  string ( $\partial$  - [55] 70)
  where ( $\partial$   $x$ ) = "d["@ $x$ @']"

```

```

definition varDiffs :: string set
  where varDiffs = { $y$ .  $\exists$   $x$ .  $y = \partial$   $x$ }

```

```

proposition vdiff-inj: ( $\partial$   $x$ ) = ( $\partial$   $y$ )  $\implies$   $x = y$ 
  by(simp add: vdiff-def)

```

```

proposition vdiff-noFixPoints:  $x \neq (\partial$   $x)$ 
  by(simp add: vdiff-def)

```

```

lemma varDiffsI:  $x = (\partial$   $z)$   $\implies$   $x \in$  varDiffs
  by(simp add: varDiffs-def vdiff-def)

```

```

lemma varDiffsE:
  assumes  $x \in$  varDiffs
  obtains  $y$  where  $x =$  "d["@ $y$ @']"
  using assms unfolding varDiffs-def vdiff-def by auto

```

```

proposition vdiff-invarDiffs: ( $\partial$   $x$ )  $\in$  varDiffs
  by (simp add: varDiffsI)

```

#### (primed) dSolve preliminaries

This subsection is to define a function that takes a system of ODEs (expressed as a list  $xfList$ ), a presumed solution  $uInput = [u_1, \dots, u_n]$ , a state  $s$  and a time  $t$ , and outputs the induced flow  $sol\ s[xfList \leftarrow uInput]\ t$ .

```

abbreviation varDiffs-to-zero :: real store  $\Rightarrow$  real store (sol)
  where sol  $a \equiv$  (override-on  $a$  ( $\lambda$   $x$ . 0) varDiffs)

```

```

proposition varDiffs-to-zero-vdiff[simp]: (sol  $s$ ) ( $\partial$   $x$ ) = 0
  apply(simp add: override-on-def varDiffs-def)
  by auto

```

```

proposition varDiffs-to-zero-beginning[simp]: take 2  $x \neq$  "d["  $\implies$  (sol  $s$ )  $x = s$   $x$ 
  apply(simp add: varDiffs-def override-on-def vdiff-def)
  by fastforce

```

— Next, for each entry of the input-list, we update the state using said entry.

```

definition vderiv-of  $f$   $S =$  (SOME  $f'$ . (f has-vderiv-on  $f'$ )  $S$ )

```

```

primrec state-list-upd :: ((real  $\Rightarrow$  real store  $\Rightarrow$  real)  $\times$  string  $\times$  (real store  $\Rightarrow$  real)) list  $\Rightarrow$ 
real  $\Rightarrow$  real store  $\Rightarrow$  real store where
  state-list-upd []  $t$   $s = s$  |
  state-list-upd ( $uxf \#$  tail)  $t$   $s =$  (state-list-upd tail  $t$   $s$ )
    (( $\pi_1$  ( $\pi_2$   $uxf$ )) := ( $\pi_1$   $uxf$ )  $t$   $s$ ,

```



$$\partial (\pi_1 (\pi_2 \text{ uxf})) := (\text{if } t = 0 \text{ then } (\pi_2 (\pi_2 \text{ uxf})) \text{ } s \\ \text{else } \text{vderiv-of } (\lambda r. (\pi_1 \text{ uxf}) \text{ } r \text{ } s) \{0 <..< (2 *_R t)\} \text{ } t))$$

**abbreviation** *state-list-cross-upd* :: *real store*  $\Rightarrow$  (*string*  $\times$  (*real store*  $\Rightarrow$  *real*)) *list*  $\Rightarrow$  (*real*  $\Rightarrow$  *real store*  $\Rightarrow$  *real*) *list*  $\Rightarrow$  *real*  $\Rightarrow$  (*char list*  $\Rightarrow$  *real*) ( $[-\leftarrow-]$  - [64,64,64] 63)  
**where**  $s[\text{xfList} \leftarrow \text{uInput}] \text{ } t \equiv \text{state-list-upd } (\text{uInput} \otimes \text{xfList}) \text{ } t \text{ } s$

**proposition** *state-list-cross-upd-empty*[*simp*]: ( $s[\leftarrow \text{list}] \text{ } t$ ) = *s*  
**by**(*induction list*, *simp-all*)

**lemma** *inductive-state-list-cross-upd-its-vars*:

**assumes** *distHyp*:*distinct* (map  $\pi_1$  ((*y*, *g*) # *xfTail*))  
**and** *varHyp*: $\forall \text{xf} \in \text{set}((\text{y}, \text{g}) \# \text{xfTail}). \pi_1 \text{xf} \notin \text{varDiffs}$   
**and** *indHyp*: $(\text{u}, \text{x}, \text{f}) \in \text{set}(\text{utail} \otimes \text{xfTail}) \implies (s[\text{xfTail} \leftarrow \text{utail}] \text{ } t) \text{ } \text{x} = \text{u } t \text{ } s$   
**and** *disjHyp*: $(\text{u}, \text{x}, \text{f}) = (\text{v}, \text{y}, \text{g}) \vee (\text{u}, \text{x}, \text{f}) \in \text{set}(\text{utail} \otimes \text{xfTail})$   
**shows** ( $s[(\text{y}, \text{g}) \# \text{xfTail} \leftarrow \text{v} \# \text{utail}] \text{ } t$ )  $\text{x} = \text{u } t \text{ } s$   
**using** *disjHyp* **proof**  
**assume**  $(\text{u}, \text{x}, \text{f}) = (\text{v}, \text{y}, \text{g})$   
**hence** ( $s[(\text{y}, \text{g}) \# \text{xfTail} \leftarrow \text{v} \# \text{utail}] \text{ } t$ )  $\text{x} = ((s[\text{xfTail} \leftarrow \text{utail}] \text{ } t)(\text{x} := \text{u } t \text{ } s,$   
 $\partial \text{x} := \text{if } t = 0 \text{ then } \text{f } s \text{ else } \text{vderiv-of } (\lambda r. \text{u } r \text{ } s) \{0 <..< (2 *_R t)\} \text{ } t)) \text{ } \text{x}$   
**by** *simp*  
**also have** ... =  $\text{u } t \text{ } s$   
**by** (*simp add: vdiff-def*)  
**ultimately show** *?thesis*  
**by** *simp*

**next**

**assume** *yTailHyp*: $(\text{u}, \text{x}, \text{f}) \in \text{set}(\text{utail} \otimes \text{xfTail})$   
**from** *this* **and** *indHyp* **have**  $\exists (s[\text{xfTail} \leftarrow \text{utail}] \text{ } t) \text{ } \text{x} = \text{u } t \text{ } s$   
**by** *fastforce*  
**from** *yTailHyp* **and** *distHyp* **have**  $2:\text{y} \neq \text{x}$  **using** *set-zip-left-rightD*  
**by** *force*  
**from** *yTailHyp* **and** *varHyp* **have**  $1:\text{x} \neq \partial \text{y}$   
**using** *set-zip-left-rightD vdiff-invarDiffs* **by** *fastforce*  
**from** 1 **and** 2 **have** ( $s[(\text{y}, \text{g}) \# \text{xfTail} \leftarrow \text{v} \# \text{utail}] \text{ } t$ )  $\text{x} = (s[\text{xfTail} \leftarrow \text{utail}] \text{ } t) \text{ } \text{x}$   
**by** *simp*  
**thus** *?thesis* **using** 3  
**by** *simp*

**qed**

**theorem** *state-list-cross-upd-its-vars*:

**assumes** *distinctHyp*:*distinct* (map  $\pi_1$  *xfList*)  
**and** *lengthHyp*:*length* *xfList* = *length* *uInput*  
**and** *varsHyp*: $\forall \text{xf} \in \text{set } \text{xfList}. \pi_1 \text{xf} \notin \text{varDiffs}$   
**and** *its-var*: $(\text{u}, \text{x}, \text{f}) \in \text{set}(\text{uInput} \otimes \text{xfList})$   
**shows** ( $s[\text{xfList} \leftarrow \text{uInput}] \text{ } t$ )  $\text{x} = \text{u } t \text{ } s$   
**using** *assms* **apply**(*induct* *xfList* *uInput* *arbitrary*: *x* *rule*: *list-induct2'*, *simp*, *simp*, *simp*)  
**by**(*clarify*, *rule* *inductive-state-list-cross-upd-its-vars*, *simp-all*)

**lemma** *override-on-upd*: $\text{x} \in X \implies (\text{override-on } f \text{ } g \text{ } X)(\text{x} := z) = (\text{override-on } f \text{ } (g(\text{x} := z))) \text{ } X$   
**by** (*rule ext*, *simp add: override-on-def*)

**lemma** *inductive-state-list-cross-upd-its-dvars*:

**assumes**  $\exists g. (s[\text{xfTail} \leftarrow \text{uTail}] \text{ } 0) = \text{override-on } s \text{ } g \text{ } \text{varDiffs}$   
**and**  $\forall \text{xf} \in \text{set}(\text{xf} \# \text{xfTail}). \pi_1 \text{xf} \notin \text{varDiffs}$   
**and**  $\forall \text{uxf} \in \text{set}(\text{u} \# \text{uTail} \otimes \text{xf} \# \text{xfTail}). \pi_1 \text{uxf} \text{ } 0 \text{ } s = s(\pi_1 (\pi_2 \text{uxf}))$   
**shows**  $\exists g. (s[\text{xf} \# \text{xfTail} \leftarrow \text{u} \# \text{uTail}] \text{ } 0) = \text{override-on } s \text{ } g \text{ } \text{varDiffs}$

**proof**–

**let** *?gLHS* = ( $s[(\text{xf} \# \text{xfTail}) \leftarrow (\text{u} \# \text{uTail})] \text{ } 0$ )  
**have** *observ*: $\partial (\pi_1 \text{xf}) \in \text{varDiffs}$  **by** (*auto simp: varDiffs-def*)  
**from** *assms*(1) **obtain** *g* **where** ( $s[\text{xfTail} \leftarrow \text{uTail}] \text{ } 0$ ) = *override-on* *s* *g* *varDiffs* **by** *force*

then have  $?gLHS = (override-on\ s\ g\ varDiffs)(\pi_1\ xf := u\ 0\ s, \partial(\pi_1\ xf) := \pi_2\ xf\ s)$  by *simp*  
 also have  $\dots = (override-on\ s\ g\ varDiffs)(\partial(\pi_1\ xf) := \pi_2\ xf\ s)$   
 using *override-on-def varDiffs-def assms* by *auto*  
 also have  $\dots = (override-on\ s\ (g(\partial(\pi_1\ xf) := \pi_2\ xf\ s))\ varDiffs)$   
 using *observ* and *override-on-upd* by *force*  
 ultimately show  $?thesis$  by *auto*  
 qed

**theorem** *state-list-cross-upd-its-dvars*:

assumes  $lengthHyp: length\ xfList = length\ uInput$   
 and  $varsHyp: \forall\ xf \in set\ xfList. \pi_1\ xf \notin varDiffs$   
 and  $solHyp1: \forall\ uxf \in set\ (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ s = s\ (\pi_1\ (\pi_2\ uxf))$   
 shows  $\exists\ g. (s[xfList \leftarrow uInput]\ 0) = (override-on\ s\ g\ varDiffs)$   
 using *assms* **proof**(*induct xfList uInput* rule: *list-induct2'*)  
 case 1  
 have  $(s[\square \leftarrow \square]\ 0) = override-on\ s\ s\ varDiffs$   
 unfolding *override-on-def* by *simp*  
 thus  $?case$  by *metis*  
 next  
 case (2  $xf\ xfTail$ )  
 have  $(s[(xf \# xfTail) \leftarrow \square]\ 0) = override-on\ s\ s\ varDiffs$   
 unfolding *override-on-def* by *simp*  
 thus  $?case$  by *metis*  
 next  
 case (3  $u\ utail$ )  
 have  $(s[\square \leftarrow utail]\ 0) = override-on\ s\ s\ varDiffs$   
 unfolding *override-on-def* by *simp*  
 thus  $?case$  by *force*  
 next  
 case (4  $xf\ xfTail\ u\ uTail$ )  
 then have  $\exists\ g. (s[xfTail \leftarrow uTail]\ 0) = override-on\ s\ g\ varDiffs$  by *simp*  
 thus  $?case$  using *inductive-state-list-cross-upd-its-dvars 4.prem*s by *blast*  
 qed

**lemma** *vderiv-unique-within-open-interval*:

assumes  $(f\ has-vderiv-on\ f')\ \{0 < .. < t\}$  and  $t > 0$   
 and  $(f\ has-vderiv-on\ f'')\ \{0 < .. < t\}$  and  $tauHyp: \tau \in \{0 < .. < t\}$   
 shows  $f'\ \tau = f''\ \tau$   
 using *assms* **apply**(*simp add: has-vderiv-on-def has-vector-derivative-def*)  
 using *frechet-derivative-unique-within-open-interval* by (*metis* *box-real(1)* *scaleR-one* *tauHyp*)

**lemma** *has-vderiv-on-cong-open-interval*:

assumes  $gHyp: \forall\ \tau > 0. f\ \tau = g\ \tau$  and  $tHyp: t > 0$   
 and  $fHyp: (f\ has-vderiv-on\ f')\ \{0 < .. < t\}$   
 shows  $(g\ has-vderiv-on\ f')\ \{0 < .. < t\}$

**proof**–

from  $gHyp$  have  $\bigwedge \tau. \tau \in \{0 < .. < t\} \implies f\ \tau = g\ \tau$  using  $tHyp$  by *force*  
 hence  $eqDs: (f\ has-vderiv-on\ f')\ \{0 < .. < t\} = (g\ has-vderiv-on\ f')\ \{0 < .. < t\}$   
**apply**(*rule-tac has-vderiv-on-cong*) by *auto*  
 thus  $(g\ has-vderiv-on\ f')\ \{0 < .. < t\}$  using  $eqDs\ fHyp$  by *simp*

qed

**lemma** *closed-vderiv-on-cong-to-open-vderiv*:

assumes  $gHyp: \forall\ \tau > 0. f\ \tau = g\ \tau$   
 and  $fHyp: \forall\ t \geq 0. (f\ has-vderiv-on\ f')\ \{0..t\}$   
 and  $tHyp: t > 0$  and  $cHyp: c > 1$   
 shows  $vderiv-of\ g\ \{0 < .. < (c *_{\mathbb{R}} t)\}\ t = f'\ t$

**proof**–

have  $ctHyp: c \cdot t > 0$  using  $tHyp$  and  $cHyp$  by *auto*  
 from  $fHyp$  have  $(f\ has-vderiv-on\ f')\ \{0 < .. < c \cdot t\}$  using *has-vderiv-on-subset*

by (metis greaterThanLessThan-subseteq-atLeastAtMost-iff less-eq-real-def)  
 then have derivHyp:(g has-vderiv-on f') {0<.. $c \cdot t$ }  
 using gHyp ctHyp and has-vderiv-on-cong-open-interval by blast  
 hence f'Hyp: $\forall f''.$  (g has-vderiv-on f'') {0<.. $c \cdot t$ }  $\longrightarrow (\forall \tau \in \{0<.. $c \cdot t$ \}. f' \tau = f'' \tau)$   
 using vderiv-unique-within-open-interval ctHyp by blast  
 also have (g has-vderiv-on (vderiv-of g {0<.. $(c *_R t)$ \})) {0<.. $c \cdot t$ }  
 by(simp add: vderiv-of-def, metis derivHyp someI-ex)  
 ultimately show vderiv-of g {0<.. $c *_R t$ }  $t = f' t$  using tHyp cHyp by force  
 qed

lemma vderiv-of-to-sol-its-vars:

assumes distinctHyp:distinct (map  $\pi_1$  xfList)  
 and lengthHyp:length xfList = length uInput  
 and varsHyp: $\forall xf \in \text{set } xfList. \pi_1 xf \notin \text{varDiffs}$   
 and solHyp2: $\forall t \geq 0. ((\lambda \tau. (\text{sol } s[xfList \leftarrow uInput] \tau)) x)$   
 has-vderiv-on ( $\lambda \tau. f (\text{sol } s[xfList \leftarrow uInput] \tau))$ ) {0.. $t$ }  
 and tHyp:  $t > 0$  and uxfHyp:(u, x, f)  $\in \text{set } (uInput \otimes xfList)$   
 shows vderiv-of ( $\lambda \tau. u \tau (\text{sol } s)$ ) {0<.. $(2 *_R t)$ }  $t = f (\text{sol } s[xfList \leftarrow uInput] t)$   
 apply(rule-tac f=( $\lambda \tau. (\text{sol } s[xfList \leftarrow uInput] \tau) x$ ) in closed-vderiv-on-cong-to-open-vderiv)  
 subgoal using assms and state-list-cross-upd-its-vars by metis  
 by(simp-all add: solHyp2 tHyp)

lemma inductive-to-sol-zero-its-dvars:

assumes eqFuncs: $\forall s. \forall g. \forall xf \in \text{set } ((x, f) \# xfs). \pi_2 xf (\text{override-on } s g \text{ varDiffs}) = \pi_2 xf s$   
 and eqLengths:length ((x, f) # xfs) = length (u # us)  
 and distinct:distinct (map  $\pi_1$  ((x, f) # xfs))  
 and vars: $\forall xf \in \text{set } ((x, f) \# xfs). \pi_1 xf \notin \text{varDiffs}$   
 and solHyp1: $\forall uxf \in \text{set } ((u \# us) \otimes ((x, f) \# xfs)). \pi_1 uxf 0 (\text{sol } s) = \text{sol } s (\pi_1 (\pi_2 uxf))$   
 and disjHyp:(y, g) = (x, f)  $\vee (y, g) \in \text{set } xfs$   
 and indHyp:(y, g)  $\in \text{set } xfs \implies (\text{sol } s[xfs \leftarrow us] 0) (\partial y) = g (\text{sol } s[xfs \leftarrow us] 0)$   
 shows (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0) ( $\partial y$ ) = g (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0)

proof—

from assms obtain h1 where h1Def:(sol s[((x, f) # xfs)  $\leftarrow$  (u # us)] 0) =  
 (override-on (sol s) h1 varDiffs) using state-list-cross-upd-its-dvars by blast  
 from disjHyp show (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0) ( $\partial y$ ) = g (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0)  
 proof

assume eqHeads:(y, g) = (x, f)  
 then have g (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0) = f (sol s) using h1Def eqFuncs by simp  
 also have ... = (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0) ( $\partial y$ ) using eqHeads by auto  
 ultimately show ?thesis by linarith

next

assume tailHyp:(y, g)  $\in \text{set } xfs$   
 then have  $y \neq x$  using distinct set-zip-left-rightD by force  
 hence  $\partial x \neq \partial y$  by(simp add: vdiff-def)  
 have  $x \neq \partial y$  using vars vdiff-invarDiffs by auto  
 obtain h2 where h2Def:(sol s[xfs  $\leftarrow$  us] 0) = override-on (sol s) h2 varDiffs  
 using state-list-cross-upd-its-dvars eqLengths distinct vars and solHyp1 by force  
 have (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0) ( $\partial y$ ) = g (sol s[xfs  $\leftarrow$  us] 0)  
 using tailHyp indHyp ( $x \neq \partial y$ ) and ( $\partial x \neq \partial y$ ) by simp  
 also have ... = g (override-on (sol s) h2 varDiffs) using h2Def by simp  
 also have ... = g (sol s) using eqFuncs and tailHyp by force  
 also have ... = g (sol s[(x, f) # xfs  $\leftarrow$  u # us] 0)  
 using eqFuncs h1Def tailHyp and eq-snd-iff by fastforce  
 ultimately show ?thesis by simp

qed

qed

lemma to-sol-zero-its-dvars:

assumes funcsHyp: $\forall s. \forall g. \forall xf \in \text{set } xfList. \pi_2 xf (\text{override-on } s g \text{ varDiffs}) = \pi_2 xf s$   
 and distinctHyp:distinct (map  $\pi_1$  xfList)

**and**  $\text{lengthHyp}:\text{length } \text{xfList} = \text{length } \text{uInput}$   
**and**  $\text{varsHyp}:\forall \text{xf} \in \text{set } \text{xfList}. \pi_1 \text{xf} \notin \text{varDiffs}$   
**and**  $\text{solHyp1}:\forall \text{uxf} \in \text{set } (\text{uInput} \otimes \text{xfList}). (\pi_1 \text{uxf}) 0 (\text{sol } s) = (\text{sol } s) (\pi_1 (\pi_2 \text{uxf}))$   
**and**  $\text{ygHyp}:(y, g) \in \text{set } \text{xfList}$   
**shows**  $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] 0)(\partial y) = g (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] 0)$   
**using**  $\text{assms}$  **apply**( $\text{induct } \text{xfList } \text{uInput}$  rule:  $\text{list-induct2'}$ ,  $\text{simp}$ ,  $\text{simp}$ ,  $\text{simp}$ ,  $\text{clarify}$ )  
**by**(rule  $\text{inductive-to-sol-zero-its-dvars}$ ,  $\text{simp-all}$ )

**lemma**  $\text{inductive-to-sol-greater-than-zero-its-dvars}$ :

**assumes**  $\text{lengthHyp}:\text{length } ((y, g) \# \text{xf}) = \text{length } (v \# \text{vs})$   
**and**  $\text{distHyp}:\text{distinct } (\text{map } \pi_1 ((y, g) \# \text{xf}))$   
**and**  $\text{varHyp}:\forall \text{xf} \in \text{set } ((y, g) \# \text{xf}). \pi_1 \text{xf} \notin \text{varDiffs}$   
**and**  $\text{indHyp}:(u, x, f) \in \text{set } (\text{vs} \otimes \text{xf}) \implies (s[\text{xf} \leftarrow \text{vs}] t)(\partial x) = \text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < 2 *_{\text{R}} t\} t$   
**and**  $\text{disjHyp}:(v, y, g) = (u, x, f) \vee (u, x, f) \in \text{set } (\text{vs} \otimes \text{xf})$  **and**  $\text{tHyp}:t > 0$   
**shows**  $(s[(y, g) \# \text{xf} \leftarrow v \# \text{vs}] t)(\partial x) = \text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < 2 *_{\text{R}} t\} t$

**proof**–

**let**  $?lhs = ((s[\text{xf} \leftarrow \text{vs}] t)(y := v \text{ t } s, \partial y := \text{vderiv-of } (\lambda r. v \text{ r } s) \{0 < \dots < (2 \cdot t)\} t))(\partial x)$   
**let**  $?rhs = \text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < (2 \cdot t)\} t$   
**have**  $(s[(y, g) \# \text{xf} \leftarrow v \# \text{vs}] t)(\partial x) = ?lhs$  **using**  $\text{tHyp}$  **by**  $\text{simp}$   
**also have**  $\text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < 2 *_{\text{R}} t\} t = ?rhs$  **by**  $\text{simp}$   
**ultimately have**  $\text{obs}:\text{thesis} = (?lhs = ?rhs)$  **by**  $\text{simp}$   
**from**  $\text{disjHyp}$  **have**  $?lhs = ?rhs$

**proof**

**assume**  $\text{uxfEq}:(v, y, g) = (u, x, f)$   
**then have**  $?lhs = \text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < (2 \cdot t)\} t$  **by**  $\text{simp}$   
**also have**  $\text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < (2 \cdot t)\} t = ?rhs$  **using**  $\text{uxfEq}$  **by**  $\text{simp}$   
**ultimately show**  $?lhs = ?rhs$  **by**  $\text{simp}$

**next**

**assume**  $\text{sygTail}:(u, x, f) \in \text{set } (\text{vs} \otimes \text{xf})$   
**from this have**  $y \neq x$  **using**  $\text{distHyp}$   $\text{set-zip-left-rightD}$  **by**  $\text{force}$   
**hence**  $\partial x \neq \partial y$  **by**( $\text{simp}$   $\text{add: vdiff-def}$ )  
**have**  $y \neq \partial x$  **using**  $\text{varHyp}$  **using**  $\text{vdiff-invarDiffs}$  **by**  $\text{auto}$   
**then have**  $?lhs = (s[\text{xf} \leftarrow \text{vs}] t)(\partial x)$  **using**  $\langle y \neq \partial x \rangle$  **and**  $\langle \partial x \neq \partial y \rangle$  **by**  $\text{simp}$   
**also have**  $(s[\text{xf} \leftarrow \text{vs}] t)(\partial x) = ?rhs$  **using**  $\text{indHyp}$   $\text{sygTail}$  **by**  $\text{simp}$   
**ultimately show**  $?lhs = ?rhs$  **by**  $\text{simp}$

**qed**

**from this and obs show**  $\text{thesis}$  **by**  $\text{simp}$

**qed**

**lemma**  $\text{to-sol-greater-than-zero-its-dvars}$ :

**assumes**  $\text{distinctHyp}:\text{distinct } (\text{map } \pi_1 \text{xfList})$   
**and**  $\text{lengthHyp}:\text{length } \text{xfList} = \text{length } \text{uInput}$   
**and**  $\text{varsHyp}:\forall \text{xf} \in \text{set } \text{xfList}. \pi_1 \text{xf} \notin \text{varDiffs}$   
**and**  $\text{uxfHyp}:(u, x, f) \in \text{set } (\text{uInput} \otimes \text{xfList})$  **and**  $\text{tHyp}:t > 0$   
**shows**  $(s[\text{xfList} \leftarrow \text{uInput}] t)(\partial x) = \text{vderiv-of } (\lambda r. u \text{ r } s) \{0 < \dots < (2 *_{\text{R}} t)\} t$   
**using**  $\text{assms}$  **apply**( $\text{induct } \text{xfList } \text{uInput}$  rule:  $\text{list-induct2'}$ ,  $\text{simp}$ ,  $\text{simp}$ ,  $\text{simp}$ ,  $\text{clarify}$ )  
**by**(rule-tac  $f=f$  **in**  $\text{inductive-to-sol-greater-than-zero-its-dvars}$ ,  $\text{auto}$ )

## dInv preliminaries

Here, we introduce syntactic notation to talk about differential invariants.

**no-notation**  $\text{Antidomain-Semiring}.\text{antidomain-left-monoid-class}.\text{am-add-op}$  (**infixl**  $\oplus$  65)

**no-notation**  $\text{Dioid.times-class.opp-mult}$  (**infixl**  $\odot$  70)

**no-notation**  $\text{Lattices.inf-class.inf}$  (**infixl**  $\sqcap$  70)

**no-notation**  $\text{Lattices.sup-class.sup}$  (**infixl**  $\sqcup$  65)

**datatype**  $\text{trms} = \text{Const } \text{real } (t_C - [54] 70) \mid \text{Var } \text{string } (t_V - [54] 70) \mid$

$\text{Mns } \text{trms} (\ominus - [54] 65) \mid \text{Sum } \text{trms } \text{trms}$  (**infixl**  $\oplus$  65)  $\mid$

$\text{Mult } \text{trms } \text{trms}$  (**infixl**  $\odot$  68)

**term** *test-monoid-class.n-add-op*

**primrec** *tval* :: *trms*  $\Rightarrow$  (*real store*  $\Rightarrow$  *real*) ((1  $\llbracket$  -  $\rrbracket_t$ )) **where**

$$\begin{aligned} \llbracket t_C r \rrbracket_t &= (\lambda s. r)| \\ \llbracket t_V x \rrbracket_t &= (\lambda s. s x)| \\ \llbracket \ominus \vartheta \rrbracket_t &= (\lambda s. - (\llbracket \vartheta \rrbracket_t s))| \\ \llbracket \vartheta \oplus \eta \rrbracket_t &= (\lambda s. (\llbracket \vartheta \rrbracket_t s) + (\llbracket \eta \rrbracket_t s))| \\ \llbracket \vartheta \odot \eta \rrbracket_t &= (\lambda s. (\llbracket \vartheta \rrbracket_t s) \cdot (\llbracket \eta \rrbracket_t s)) \end{aligned}$$

**datatype** *props* = *Eq trms trms* (**infixr**  $\doteq$  60) | *Less trms trms* (**infixr**  $\prec$  62) |  
*Leq trms trms* (**infixr**  $\preceq$  61) | *And props props* (**infixl**  $\sqcap$  63) |  
*Or props props* (**infixl**  $\sqcup$  64)

**primrec** *pval* :: *props*  $\Rightarrow$  (*real store*  $\Rightarrow$  *bool*) ((1  $\llbracket$  -  $\rrbracket_P$ )) **where**

$$\begin{aligned} \llbracket \vartheta \doteq \eta \rrbracket_P &= (\lambda s. (\llbracket \vartheta \rrbracket_t s) = (\llbracket \eta \rrbracket_t s))| \\ \llbracket \vartheta \prec \eta \rrbracket_P &= (\lambda s. (\llbracket \vartheta \rrbracket_t s) < (\llbracket \eta \rrbracket_t s))| \\ \llbracket \vartheta \preceq \eta \rrbracket_P &= (\lambda s. (\llbracket \vartheta \rrbracket_t s) \leq (\llbracket \eta \rrbracket_t s))| \\ \llbracket \varphi \sqcap \psi \rrbracket_P &= (\lambda s. (\llbracket \varphi \rrbracket_P s) \wedge (\llbracket \psi \rrbracket_P s))| \\ \llbracket \varphi \sqcup \psi \rrbracket_P &= (\lambda s. (\llbracket \varphi \rrbracket_P s) \vee (\llbracket \psi \rrbracket_P s)) \end{aligned}$$

**primrec** *tdiff* :: *trms*  $\Rightarrow$  *trms* ( $\partial_t$  - [54] 70) **where**

$$\begin{aligned} (\partial_t t_C r) &= t_C 0| \\ (\partial_t t_V x) &= t_V (\partial x)| \\ (\partial_t \ominus \vartheta) &= \ominus (\partial_t \vartheta)| \\ (\partial_t (\vartheta \oplus \eta)) &= (\partial_t \vartheta) \oplus (\partial_t \eta)| \\ (\partial_t (\vartheta \odot \eta)) &= ((\partial_t \vartheta) \odot \eta) \oplus (\vartheta \odot (\partial_t \eta)) \end{aligned}$$

**primrec** *pdiff* :: *props*  $\Rightarrow$  *props* ( $\partial_P$  - [54] 70) **where**

$$\begin{aligned} (\partial_P (\vartheta \doteq \eta)) &= ((\partial_t \vartheta) \doteq (\partial_t \eta))| \\ (\partial_P (\vartheta \prec \eta)) &= ((\partial_t \vartheta) \prec (\partial_t \eta))| \\ (\partial_P (\vartheta \preceq \eta)) &= ((\partial_t \vartheta) \preceq (\partial_t \eta))| \\ (\partial_P (\varphi \sqcap \psi)) &= (\partial_P \varphi) \sqcap (\partial_P \psi)| \\ (\partial_P (\varphi \sqcup \psi)) &= (\partial_P \varphi) \sqcap (\partial_P \psi) \end{aligned}$$

**primrec** *trmVars* :: *trms*  $\Rightarrow$  *string set* **where**

$$\begin{aligned} \text{trmVars } (t_C r) &= \{\}| \\ \text{trmVars } (t_V x) &= \{x\}| \\ \text{trmVars } (\ominus \vartheta) &= \text{trmVars } \vartheta| \\ \text{trmVars } (\vartheta \oplus \eta) &= \text{trmVars } \vartheta \cup \text{trmVars } \eta| \\ \text{trmVars } (\vartheta \odot \eta) &= \text{trmVars } \vartheta \cup \text{trmVars } \eta \end{aligned}$$

**fun** *substList* :: (*string*  $\times$  *trms*) *list*  $\Rightarrow$  *trms*  $\Rightarrow$  *trms* ( $\langle$  -  $\rangle$  [54] 80) **where**

$$\begin{aligned} \text{xtList } \langle t_C r \rangle &= t_C r| \\ \llbracket \langle t_V x \rangle &= t_V x| \\ ((y, \xi) \# \text{xtTail } \langle \text{Var } x \rangle) &= (\text{if } x = y \text{ then } \xi \text{ else } \text{xtTail } \langle \text{Var } x \rangle)| \\ \text{xtList } \langle \ominus \vartheta \rangle &= \ominus (\text{xtList } \langle \vartheta \rangle)| \\ \text{xtList } \langle \vartheta \oplus \eta \rangle &= (\text{xtList } \langle \vartheta \rangle) \oplus (\text{xtList } \langle \eta \rangle)| \\ \text{xtList } \langle \vartheta \odot \eta \rangle &= (\text{xtList } \langle \vartheta \rangle) \odot (\text{xtList } \langle \eta \rangle) \end{aligned}$$

**proposition** *substList-on-compl-of-varDiffs*:

**assumes** *trmVars*  $\eta \subseteq (\text{UNIV} - \text{varDiffs})$   
**and** *set* (*map*  $\pi_1$  *xtList*)  $\subseteq \text{varDiffs}$   
**shows** *xtList*  $\langle \eta \rangle = \eta$   
**using** *assms* **apply** (*induction*  $\eta$ , *simp-all* *add*: *varDiffs-def*)  
**by** (*induction* *xtList*, *auto*)

**lemma** *substList-help1*: *set* (*map*  $\pi_1$  ((*map* (*vdiff*  $\circ$   $\pi_1$ ) *xfList*)  $\otimes$  *uInput*))  $\subseteq \text{varDiffs}$   
**apply** (*induct* *xfList* *uInput* *rule*: *list-induct2'*, *simp-all* *add*: *varDiffs-def*)  
**by** *auto*

**lemma** *substList-help2*:  
**assumes**  $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$   
**shows**  $((\text{map } (\text{vdiff} \circ \pi_1) \text{xfList}) \otimes \text{uInput}) \langle \eta \rangle = \eta$   
**using** *assms substList-help1 substList-on-compl-of-varDiffs* **by** *blast*

**lemma** *substList-cross-vdiff-on-non-occurring-var*:  
**assumes**  $x \notin \text{set list1}$   
**shows**  $((\text{map } \text{vdiff } \text{list1}) \otimes \text{list2}) \langle t_V (\partial x) \rangle = t_V (\partial x)$   
**using** *assms apply(induct list1 list2 rule: list-induct2', simp, simp, clarsimp)*  
**by** *(simp add: vdiff-def)*

**primrec** *propVars* ::  $\text{props} \Rightarrow \text{string set}$  **where**  
 $\text{propVars } (\vartheta \doteq \eta) = \text{trmVars } \vartheta \cup \text{trmVars } \eta$   
 $\text{propVars } (\vartheta \prec \eta) = \text{trmVars } \vartheta \cup \text{trmVars } \eta$   
 $\text{propVars } (\vartheta \preceq \eta) = \text{trmVars } \vartheta \cup \text{trmVars } \eta$   
 $\text{propVars } (\varphi \sqcap \psi) = \text{propVars } \varphi \cup \text{propVars } \psi$   
 $\text{propVars } (\varphi \sqcup \psi) = \text{propVars } \varphi \cup \text{propVars } \psi$

**primrec** *subspList* ::  $(\text{string} \times \text{trms}) \text{ list} \Rightarrow \text{props} \Rightarrow \text{props}$  ( $-\vdash$  [54] 80) **where**  
 $\text{xtList} \vdash \vartheta \doteq \eta \vdash = ((\text{xtList} \langle \vartheta \rangle) \doteq (\text{xtList} \langle \eta \rangle))$   
 $\text{xtList} \vdash \vartheta \prec \eta \vdash = ((\text{xtList} \langle \vartheta \rangle) \prec (\text{xtList} \langle \eta \rangle))$   
 $\text{xtList} \vdash \vartheta \preceq \eta \vdash = ((\text{xtList} \langle \vartheta \rangle) \preceq (\text{xtList} \langle \eta \rangle))$   
 $\text{xtList} \vdash \varphi \sqcap \psi \vdash = ((\text{xtList} \vdash \varphi \vdash) \sqcap (\text{xtList} \vdash \psi \vdash))$   
 $\text{xtList} \vdash \varphi \sqcup \psi \vdash = ((\text{xtList} \vdash \varphi \vdash) \sqcup (\text{xtList} \vdash \psi \vdash))$

## ODE Extras

For exemplification purposes, we compile some concrete derivatives used commonly in classical mechanics. A more general approach should be taken that generates this theorems as instantiations.

**named-theorems** *ubc-definitions definitions used in the locale unique-on-bounded-closed*

**declare** *unique-on-bounded-closed-def* [*ubc-definitions*]  
**and** *unique-on-bounded-closed-axioms-def* [*ubc-definitions*]  
**and** *unique-on-closed-def* [*ubc-definitions*]  
**and** *compact-interval-def* [*ubc-definitions*]  
**and** *compact-interval-axioms-def* [*ubc-definitions*]  
**and** *self-mapping-def* [*ubc-definitions*]  
**and** *self-mapping-axioms-def* [*ubc-definitions*]  
**and** *continuous-rhs-def* [*ubc-definitions*]  
**and** *closed-domain-def* [*ubc-definitions*]  
**and** *global-lipschitz-def* [*ubc-definitions*]  
**and** *interval-def* [*ubc-definitions*]  
**and** *nonempty-set-def* [*ubc-definitions*]  
**and** *lipschitz-on-def* [*ubc-definitions*]

**named-theorems** *poly-deriv temporal compilation of derivatives representing galilean transformations*

**named-theorems** *galilean-transform temporal compilation of vderivs representing galilean transformations*

**named-theorems** *galilean-transform-eq the equational version of galilean-transform*

**lemma** *vector-derivative-line-at-origin*:  $((\cdot) \text{ a has-vector-derivative } a) \text{ (at } x \text{ within } T)$   
**by** *(auto intro: derivative-eq-intros)*

**lemma** [*poly-deriv*]:  $((\cdot) \text{ a has-derivative } (\lambda x. x *_R a)) \text{ (at } x \text{ within } T)$   
**using** *vector-derivative-line-at-origin unfolding has-vector-derivative-def* **by** *simp*

**lemma** *quadratic-monomial-derivative*:  
 $((\lambda t::\text{real}. a \cdot t^2) \text{ has-derivative } (\lambda t. a \cdot (2 \cdot x \cdot t))) \text{ (at } x \text{ within } T)$   
**apply** *(rule-tac g'1= $\lambda t. 2 \cdot x \cdot t$  in derivative-eq-intros(6))*

**apply**(rule-tac  $f'1=\lambda t. t$  **in** derivative-eq-intros(16))  
**by** (auto intro: derivative-eq-intros)

**lemma** quadratic-monomial-derivative2:

(( $\lambda t::\text{real}. a \cdot t^2 / 2$ ) has-derivative ( $\lambda t. a \cdot x \cdot t$ )) (at  $x$  within  $T$ )  
**apply**(rule-tac  $f'1=\lambda t. a \cdot (2 \cdot x \cdot t)$  **and**  $g'1=\lambda x. 0$  **in** derivative-eq-intros(18))  
**using** quadratic-monomial-derivative **by** auto

**lemma** quadratic-monomial-vderiv[poly-deriv]:(( $\lambda t. a \cdot t^2 / 2$ ) has-vderiv-on ( $\cdot$ )  $a$ )  $T$

**apply**(simp add: has-vderiv-on-def has-vector-derivative-def, clarify)  
**using** quadratic-monomial-derivative2 **by** (simp add: mult-commute-abs)

**lemma** galilean-position[galilean-transform]:

(( $\lambda t. a \cdot t^2 / 2 + v \cdot t + x$ ) has-vderiv-on ( $\lambda t. a \cdot t + v$ ))  $T$   
**apply**(rule-tac  $f'=\lambda x. a \cdot x + v$  **and**  $g'1=\lambda x. 0$  **in** derivative-intros(189))  
**apply**(rule-tac  $f'1=\lambda x. a \cdot x$  **and**  $g'1=\lambda x. v$  **in** derivative-intros(189))  
**using** poly-deriv(2) **by**(auto intro: derivative-intros)

**lemma** [poly-deriv]:

$t \in T \implies ((\lambda \tau. a \cdot \tau^2 / 2 + v \cdot \tau + x)$  has-derivative ( $\lambda x. x *_R (a \cdot t + v)$ )) (at  $t$  within  $T$ )  
**using** galilean-position **unfolding** has-vderiv-on-def has-vector-derivative-def **by** simp

**lemma** [galilean-transform-eq]:

$t > 0 \implies \text{vderiv-of } (\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \{0 <..< 2 \cdot t\} t = a \cdot t + v$

**proof**–

**let**  $?f = \text{vderiv-of } (\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \{0 <..< 2 \cdot t\}$   
**assume**  $t > 0$  **hence**  $t \in \{0 <..< 2 \cdot t\}$  **by** auto  
**have**  $\exists f. ((\lambda t. a \cdot t^2 / 2 + v \cdot t + x)$  has-vderiv-on  $f$ )  $\{0 <..< 2 \cdot t\}$   
**using** galilean-position **by** blast  
**hence**  $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x)$  has-vderiv-on  $?f$ )  $\{0 <..< 2 \cdot t\}$   
**unfolding** vderiv-of-def **by** (metis (mono-tags, lifting) someI-ex)  
**also have**  $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x)$  has-vderiv-on ( $\lambda t. a \cdot t + v$ ))  $\{0 <..< 2 \cdot t\}$   
**using** galilean-position **by** simp  
**ultimately show**  $(\text{vderiv-of } (\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \{0 <..< 2 \cdot t\}) t = a \cdot t + v$   
**apply**(rule-tac  $f'=?f$  **and**  $\tau=t$  **and**  $t=2 \cdot t$  **in** vderiv-unique-within-open-interval)  
**using**  $\{t \in \{0 <..< 2 \cdot t\}\}$  **by** auto

**qed**

**lemma**  $t > 0 \implies \text{vderiv-of } (\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \{0 <..< 2 \cdot t\} t = a \cdot t + v$

**unfolding** vderiv-of-def **apply**(subst someI-equality[of - ( $\lambda t. a \cdot t + v$ )])  
**apply**(rule-tac  $a=\lambda t. a \cdot t + v$  **in** ex1I)  
**apply**(simp-all add: galilean-position)  
**apply**(rule ext, rename-tac  $f \tau$ )  
**apply**(rule-tac  $f=\lambda t. a \cdot t^2 / 2 + v \cdot t + x$  **and**  $t=2 \cdot t$  **and**  $f'=f$  **in** vderiv-unique-within-open-interval)  
**apply**(simp-all add: galilean-position)  
**oops**

**lemma** galilean-velocity[galilean-transform]:(( $\lambda r. a \cdot r + v$ ) has-vderiv-on ( $\lambda t. a$ ))  $T$

**apply**(rule-tac  $f'1=\lambda x. a$  **and**  $g'1=\lambda x. 0$  **in** derivative-intros(189))  
**unfolding** has-vderiv-on-def **by**(auto intro: derivative-eq-intros)

**lemma** [galilean-transform-eq]:

$t > 0 \implies \text{vderiv-of } (\lambda r. a \cdot r + v) \{0 <..< 2 \cdot t\} t = a$

**proof**–

**let**  $?f = \text{vderiv-of } (\lambda r. a \cdot r + v) \{0 <..< 2 \cdot t\}$   
**assume**  $t > 0$  **hence**  $t \in \{0 <..< 2 \cdot t\}$  **by** auto  
**have**  $\exists f. ((\lambda r. a \cdot r + v)$  has-vderiv-on  $f$ )  $\{0 <..< 2 \cdot t\}$   
**using** galilean-velocity **by** blast  
**hence**  $((\lambda r. a \cdot r + v)$  has-vderiv-on  $?f$ )  $\{0 <..< 2 \cdot t\}$

**unfolding** *vderiv-of-def* **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)  
**also have**  $((\lambda r. a \cdot r + v) \text{ has-}v\text{deriv-on } (\lambda t. a)) \{0 <..< 2 \cdot t\}$   
**using** *galilean-velocity* **by** *simp*  
**ultimately show**  $(v\text{deriv-of } (\lambda r. a \cdot r + v) \{0 <..< 2 \cdot t\}) t = a$   
**apply**(*rule-tac*  $f'=?f$  **and**  $\tau=t$  **and**  $t=2 \cdot t$  **in** *vderiv-unique-within-open-interval*)  
**using**  $\langle t \in \{0 <..< 2 \cdot t\} \rangle$  **by** *auto*  
**qed**

**lemma** [*galilean-transform*]:  
 $((\lambda t. v \cdot t - a \cdot t^2 / 2 + x) \text{ has-}v\text{deriv-on } (\lambda x. v - a \cdot x)) \{0..t\}$   
**apply**(*subgoal-tac*  $((\lambda t. - a \cdot t^2 / 2 + v \cdot t + x) \text{ has-}v\text{deriv-on } (\lambda x. - a \cdot x + v)) \{0..t\}, \text{ simp})$   
**by**(*rule galilean-transform*)

**lemma** [*galilean-transform-eq*]:  $t > 0 \implies v\text{deriv-of } (\lambda t. v \cdot t - a \cdot t^2 / 2 + x) \{0 <..< 2 \cdot t\} t = v - a \cdot t$   
**apply**(*subgoal-tac*  $v\text{deriv-of } (\lambda t. - a \cdot t^2 / 2 + v \cdot t + x) \{0 <..< 2 \cdot t\} t = - a \cdot t + v, \text{ simp})$   
**by**(*rule galilean-transform-eq*)

**lemma** [*galilean-transform*]:  
 $((\lambda t. v - a \cdot t) \text{ has-}v\text{deriv-on } (\lambda x. - a)) \{0..t\}$   
**apply**(*subgoal-tac*  $((\lambda t. - a \cdot t + v) \text{ has-}v\text{deriv-on } (\lambda x. - a)) \{0..t\}, \text{ simp})$   
**by**(*rule galilean-transform*)

**lemma** [*galilean-transform-eq*]:  $t > 0 \implies v\text{deriv-of } (\lambda r. v - a \cdot r) \{0 <..< 2 \cdot t\} t = - a$   
**apply**(*subgoal-tac*  $v\text{deriv-of } (\lambda t. - a \cdot t + v) \{0 <..< 2 \cdot t\} t = - a, \text{ simp})$   
**by**(*rule galilean-transform-eq*)

**lemma** [*simp*]:  $(\lambda x. \text{case } x \text{ of } (t, x) \Rightarrow f t) = (\lambda x. (f \circ \pi_1) x)$   
**by** *auto*

**end**  
**theory** *VC-diffKAD*  
**imports** *VC-diffKAD-auxiliarities*

**begin**

### 0.19.3 Phase Space Relational Semantics

**definition** *solvesStoreIVP* ::  $(\text{real} \Rightarrow \text{real store}) \Rightarrow (\text{string} \times (\text{real store} \Rightarrow \text{real})) \text{ list} \Rightarrow \text{real store} \Rightarrow \text{bool}$   $((- \text{ solvesTheStoreIVP - withInitState -}) [70, 70, 70] 68)$

**where** *solvesStoreIVP*  $\varphi_S \text{ xfList } s \equiv$   
 — F sends *vdiffs-in-list* to *derivs*.  
 $(\forall t \geq 0. (\forall xf \in \text{set } \text{xfList}. \varphi_S t (\partial (\pi_1 xf)) = \pi_2 xf (\varphi_S t)) \wedge$   
 — F preserves the rest of the variables and F sends *derivs* of constants to 0.  
 $(\forall y. (y \notin (\pi_1 \downarrow \text{set } \text{xfList})) \cup \text{varDiffs} \longrightarrow \varphi_S t y = s y) \wedge$   
 $(y \notin (\pi_1 \downarrow \text{set } \text{xfList})) \longrightarrow \varphi_S t (\partial y) = 0)) \wedge$   
 — F solves the induced IVP.  
 $(\forall xf \in \text{set } \text{xfList}. ((\lambda t. \varphi_S t (\pi_1 xf)) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 xf) (\varphi_S t))) \{0..t\} \text{ UNIV} \wedge$   
 $\varphi_S 0 (\pi_1 xf) = s(\pi_1 xf)))$

**lemma** *solves-store-ivpI*:  
**assumes**  $\forall t \geq 0. \forall xf \in \text{set } \text{xfList}. (\varphi_S t (\partial (\pi_1 xf))) = (\pi_2 xf) (\varphi_S t)$   
**and**  $\forall t \geq 0. \forall y. y \notin (\pi_1 \downarrow \text{set } \text{xfList}) \cup \text{varDiffs} \longrightarrow \varphi_S t y = s y$   
**and**  $\forall t \geq 0. \forall y. y \notin (\pi_1 \downarrow \text{set } \text{xfList}) \longrightarrow \varphi_S t (\partial y) = 0$   
**and**  $\forall t \geq 0. \forall xf \in \text{set } \text{xfList}. ((\lambda t. \varphi_S t (\pi_1 xf)) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 xf) (\varphi_S t))) \{0..t\} \text{ UNIV}$   
**and**  $\forall xf \in \text{set } \text{xfList}. \varphi_S 0 (\pi_1 xf) = s(\pi_1 xf)$   
**shows**  $\varphi_S \text{ solvesTheStoreIVP } \text{xfList withInitState } s$   
**apply**(*simp add: solvesStoreIVP-def, safe*)  
**using** *assms apply simp-all*  
**by**(*force,force,force*)



**named-theorems** *solves-store-ivpE* elimination rules for *solvesStoreIVP*

**lemma** [*solves-store-ivpE*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**shows**  $\forall t \geq 0. \forall y. y \notin (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs} \longrightarrow \varphi_S t y = s y$   
**and**  $\forall t \geq 0. \forall y. y \notin (\pi_1(\text{set } \text{xfList})) \longrightarrow \varphi_S t (\partial y) = 0$   
**and**  $\forall t \geq 0. \forall \text{xf} \in \text{set } \text{xfList}. (\varphi_S t (\partial (\pi_1 \text{xf}))) = (\pi_2 \text{xf}) (\varphi_S t)$   
**and**  $\forall t \geq 0. \forall \text{xf} \in \text{set } \text{xfList}. ((\lambda t. \varphi_S t (\pi_1 \text{xf})) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 \text{xf}) (\varphi_S t))) \{0..t\} \text{ UNIV}$   
**and**  $\forall \text{xf} \in \text{set } \text{xfList}. \varphi_S 0 (\pi_1 \text{xf}) = s(\pi_1 \text{xf})$   
**using** *assms solvesStoreIVP-def* **by** *auto*

**lemma** [*solves-store-ivpE*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**shows**  $\forall y. y \notin \text{varDiffs} \longrightarrow \varphi_S 0 y = s y$

**proof**(*clarify*, *rename-tac* *x*)

**fix** *x* **assume**  $x \notin \text{varDiffs}$   
**from** *assms* **and** *solves-store-ivpE*(5) **have**  $x \in (\pi_1(\text{set } \text{xfList})) \implies \varphi_S 0 x = s x$  **by** *fastforce*  
**also have**  $x \notin (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs} \implies \varphi_S 0 x = s x$   
**using** *assms* **and** *solves-store-ivpE*(1) **by** *simp*  
**ultimately show**  $\varphi_S 0 x = s x$  **using**  $\langle x \notin \text{varDiffs} \rangle$  **by** *auto*

**qed**

**named-theorems** *solves-store-ivpD* computation rules for *solvesStoreIVP*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**and**  $t \geq 0$   
**and**  $y \notin (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs}$   
**shows**  $\varphi_S t y = s y$   
**using** *assms solves-store-ivpE*(1) **by** *simp*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**and**  $t \geq 0$   
**and**  $y \notin (\pi_1(\text{set } \text{xfList}))$   
**shows**  $\varphi_S t (\partial y) = 0$   
**using** *assms solves-store-ivpE*(2) **by** *simp*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**and**  $t \geq 0$   
**and**  $\text{xf} \in \text{set } \text{xfList}$   
**shows**  $(\varphi_S t (\partial (\pi_1 \text{xf}))) = (\pi_2 \text{xf}) (\varphi_S t)$   
**using** *assms solves-store-ivpE*(3) **by** *simp*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**and**  $t \geq 0$   
**and**  $\text{xf} \in \text{set } \text{xfList}$   
**shows**  $((\lambda t. \varphi_S t (\pi_1 \text{xf})) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 \text{xf}) (\varphi_S t))) \{0..t\} \text{ UNIV}$   
**using** *assms solves-store-ivpE*(4) **by** *simp*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  *solvesTheStoreIVP* *xfList* withInitState *s*  
**and**  $(x, f) \in \text{set } \text{xfList}$   
**shows**  $\varphi_S 0 x = s x$   
**using** *assms solves-store-ivpE*(5) **by** *fastforce*

**lemma** [*solves-store-ivpD*]:

**assumes**  $\varphi_S$  solvesTheStoreIVP  $xfList$  withInitState  $s$   
**and**  $y \notin \text{varDiffs}$   
**shows**  $\varphi_S \ 0 \ y = s \ y$   
**using** *assms solves-store-ivpE(6)* **by** *simp*

**definition** *guarDiffEqtn* :: (string  $\times$  (real store  $\Rightarrow$  real)) list  $\Rightarrow$  (real store pred)  $\Rightarrow$  real store rel (ODEsystem - with - [70, 70] 61)  
**where** ODEsystem  $xfList$  with  $G =$   
 $\{(s, \varphi_S \ t) \mid s \ t \ \varphi_S. \ t \geq 0 \wedge (\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r)) \wedge \text{solvesStoreIVP} \ \varphi_S \ xfList \ s\}$

**abbreviation** *vericond* :: 'a pred  $\Rightarrow$  'a rel  $\Rightarrow$  'a pred  $\Rightarrow$  bool (PRE - - POST -)  
**where**  $PRE \ P \ X \ POST \ Q \equiv \lceil P \rceil \subseteq wp \ X \ \lceil Q \rceil$

**lemma** *vericond-gevol*: (PRE  $P$  (ODEsystem  $xfList$  with  $G$ ) POST  $Q$ ) =  
 $(\forall s. \ P \ s \longrightarrow (\forall s'. \ (s, s') \in (\text{ODEsystem} \ xfList \ \text{with} \ G) \longrightarrow Q \ s'))$   
**unfolding** *wp-rel* **by** *clarsimp*

## 0.19.4 Derivation of Differential Dynamic Logic Rules

### ”Differential Weakening”

**lemma** *wlp-evol-guard:Id*  $\subseteq wp \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ \lceil G \rceil$   
**by** (*simp add: rel-aka.fbox-def rel-ad-def guarDiffEqtn-def p2r-def, force*)

**theorem** *dWeakening*:  
**assumes** *guardImpliesPost*:  $\lceil G \rceil \subseteq \lceil Q \rceil$   
**shows**  $PRE \ P \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ POST \ Q$   
**unfolding** *wp-rel guarDiffEqtn-def* **using** *assms* **by** *auto*

**theorem** *dW*:  $wp \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ \lceil Q \rceil = wp \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ \lceil \lambda s. \ G \ s \longrightarrow Q \ s \rceil$   
**unfolding** *rel-aka.fbox-def rel-ad-def guarDiffEqtn-def*  
**by** (*simp add: relcomp.simps p2r-def, fastforce*)

### ”Differential Cut”

**lemma** *all-interval-guarDiffEqtn*:  
**assumes** *solvesStoreIVP*  $\varphi_S \ xfList \ s \wedge (\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r)) \wedge 0 \leq t$   
**shows**  $\forall \ r \in \{0..t\}. \ (s, \varphi_S \ r) \in (\text{ODEsystem} \ xfList \ \text{with} \ G)$   
**unfolding** *guarDiffEqtn-def* **using** *atLeastAtMost-iff* **apply** *clarsimp*  
**apply** (*rule-tac x=r in exI, rule-tac x= $\varphi_S$  in exI*) **using** *assms* **by** *simp*

**lemma** *condAfterEvol-remainsAlongEvol*:  
**assumes** *boxDiffC*:  $(s, s) \in wp \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ \lceil C \rceil$   
**and** *FisSol*: *solvesStoreIVP*  $\varphi_S \ xfList \ s \wedge (\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r)) \wedge 0 \leq t$   
**shows**  $\forall \ r \in \{0..t\}. \ G \ (\varphi_S \ r) \wedge C \ (\varphi_S \ r)$

**proof**—

**from** *boxDiffC* **have**  $\forall \ c. \ (s, c) \in (\text{ODEsystem} \ xfList \ \text{with} \ G) \longrightarrow C \ c$   
**by** (*simp add: boxProgrPred-chrctrztn*)  
**also from** *FisSol* **have**  $\forall \ r \in \{0..t\}. \ (s, \varphi_S \ r) \in (\text{ODEsystem} \ xfList \ \text{with} \ G)$   
**using** *all-interval-guarDiffEqtn* **by** *blast*  
**ultimately show** *?thesis*  
**using** *FisSol atLeastAtMost-iff guarDiffEqtn-def* **by** *fastforce*

**qed**

**theorem** *dCut*:  
**assumes** *pBoxDiffCut*: (PRE  $P$  (ODEsystem  $xfList$  with  $G$ ) POST  $C$ )  
**assumes** *pBoxCutQ*: (PRE  $P$  (ODEsystem  $xfList$  with  $(\lambda s. \ G \ s \wedge C \ s)$ ) POST  $Q$ )  
**shows**  $PRE \ P \ (\text{ODEsystem} \ xfList \ \text{with} \ G) \ POST \ Q$   
**proof** (*clarsimp simp: wp-rel*)  
**fix**  $b \ y$  **assume**  $P \ b$  **and**  $(b, y) \in \text{ODEsystem} \ xfList \ \text{with} \ G$

**then obtain**  $\varphi_S t$  **where**  $*:solvesStoreIVP \varphi_S xflist b \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t \wedge \varphi_S t = y$   
**using** *guarDiffEqtn-def* **by** *auto*  
**hence**  $\forall r \in \{0..t\}. (b, \varphi_S r) \in (ODEsystem xflist \text{ with } G)$   
**using** *all-interval-guarDiffEqtn* **by** *blast*  
**from this and** *pBoxDiffCut* **have**  $\forall r \in \{0..t\}. C (\varphi_S r)$   
**using**  $\langle P b \rangle$  **by**(*clarsimp simp: wp-rel*)  
**then have**  $\forall r \in \{0..t\}. (b, \varphi_S r) \in (ODEsystem xflist \text{ with } (\lambda s. G s \wedge C s))$   
**using**  $*$  *all-interval-guarDiffEqtn* **by** (*metis (mono-tags, lifting)*)  
**from this and** *pBoxCutQ* **have**  $\forall r \in \{0..t\}. Q (\varphi_S r)$   
**using** *boxProgrPred-chrcrtn*  $\langle P b \rangle$  **by**(*clarsimp simp: wp-rel*)  
**thus**  $Q y$   
**using**  $*$  **by** *auto*  
**qed**

**theorem** *dC*:

**assumes**  $Id \subseteq wp (ODEsystem xflist \text{ with } G) \lceil C \rceil$   
**shows**  $wp (ODEsystem xflist \text{ with } G) \lceil Q \rceil = wp (ODEsystem xflist \text{ with } (\lambda s. G s \wedge C s)) \lceil Q \rceil$   
**proof**(*rule-tac f= $\lambda x. wp x \lceil Q \rceil$  in HOL.arg-cong, safe*)  
**fix**  $a b$  **assume**  $(a, b) \in ODEsystem xflist \text{ with } G$   
**then obtain**  $\varphi_S t$  **where**  $*:solvesStoreIVP \varphi_S xflist a \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t \wedge \varphi_S t = b$   
**using** *guarDiffEqtn-def* **by** *auto*  
**hence**  $1:\forall r \in \{0..t\}. (a, \varphi_S r) \in ODEsystem xflist \text{ with } G$   
**by** (*meson all-interval-guarDiffEqtn*)  
**from this have**  $\forall r \in \{0..t\}. C (\varphi_S r)$   
**using** *assms boxProgrPred-chrcrtn*  
**by** (*metis (no-types, hide-lams) basic-trans-rules(31) pair-in-Id-conv*)  
**thus**  $(a, b) \in ODEsystem xflist \text{ with } (\lambda s. G s \wedge C s)$   
**using**  $*$  *guarDiffEqtn-def* **by** *blast*  
**next**  
**fix**  $a b$  **assume**  $(a, b) \in ODEsystem xflist \text{ with } (\lambda s. G s \wedge C s)$   
**then show**  $(a, b) \in ODEsystem xflist \text{ with } G$   
**unfolding** *guarDiffEqtn-def* **by**(*clarsimp, rule-tac x=t in exI, rule-tac x= $\varphi_S$  in exI, simp*)  
**qed**

## Solve Differential Equation

**lemma** *prelim-dSolve*:

**assumes** *solHyp*: $(\lambda t. sol s[xflist \leftarrow uInput] t) solvesTheStoreIVP xflist withInitState s$   
**and** *uniqHyp*: $\forall X. solvesStoreIVP X xflist s \longrightarrow (\forall t \geq 0. (sol s[xflist \leftarrow uInput] t) = X t)$   
**and** *diffAssgn*: $\forall t \geq 0. G (sol s[xflist \leftarrow uInput] t) \longrightarrow Q (sol s[xflist \leftarrow uInput] t)$   
**shows**  $\forall c. (s, c) \in (ODEsystem xflist \text{ with } G) \longrightarrow Q c$   
**proof**(*clarify*)  
**fix**  $c$  **assume**  $(s, c) \in (ODEsystem xflist \text{ with } G)$   
**from this obtain**  $t::real$  **and**  $\varphi_S::real \Rightarrow real \text{ store}$   
**where**  $FHyp:t \geq 0 \wedge \varphi_S t = c \wedge solvesStoreIVP \varphi_S xflist s \wedge (\forall r \in \{0..t\}. G (\varphi_S r))$   
**using** *guarDiffEqtn-def* **by** *auto*  
**from this and** *uniqHyp* **have**  $(sol s[xflist \leftarrow uInput] t) = \varphi_S t$  **by** *blast*  
**then have**  $cHyp:c = (sol s[xflist \leftarrow uInput] t)$  **using** *FHyp* **by** *simp*  
**from this have**  $G (sol s[xflist \leftarrow uInput] t)$  **using** *FHyp* **by** *force*  
**then show**  $Q c$  **using** *diffAssgn FHyp cHyp* **by** *auto*  
**qed**

**theorem** *dS*:

**assumes** *solHyp*: $\forall s. solvesStoreIVP (\lambda t. sol s[xflist \leftarrow uInput] t) xflist s$   
**and** *uniqHyp*: $\forall s X. solvesStoreIVP X xflist s \longrightarrow (\forall t \geq 0. (sol s[xflist \leftarrow uInput] t) = X t)$   
**shows**  $wp (ODEsystem xflist \text{ with } G) \lceil Q \rceil =$   
 $\lceil \lambda s. \forall t \geq 0. (\forall r \in \{0..t\}. G (sol s[xflist \leftarrow uInput] r)) \longrightarrow Q (sol s[xflist \leftarrow uInput] t) \rceil$   
**apply**(*simp add: p2r-def, rule subset-antisym*)  
**unfolding** *guarDiffEqtn-def rel-aka.fbox-def rel-ad-def*  
**using** *solHyp* **apply**(*simp add: relcomp.simps*) **apply** *clarify*

```

apply(rule-tac  $x=x$  in  $exI$ , clarsimp)
apply(erule-tac  $x=sol\ x[xfList \leftarrow uInput]$   $t$  in  $allE$ , erule disjE)
apply(erule-tac  $x=x$  in  $allE$ , erule-tac  $x=t$  in  $allE$ )
apply(erule impE, simp, erule-tac  $x=\lambda t. sol\ x[xfList \leftarrow uInput]$   $t$  in  $allE$ )
apply(simp-all, clarify, rule-tac  $x=s$  in  $exI$ , simp add: relcomp.simps)
using uniqHyp by fastforce

```

**theorem** dSolve:

```

assumes solHyp: $\forall s. solvesStoreIVP (\lambda t. sol\ s[xfList \leftarrow uInput]\ t)\ xfList\ s$ 
  and uniqHyp: $\forall s. \forall X. solvesStoreIVP\ X\ xfList\ s \longrightarrow (\forall t \geq 0. (sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$ 
  and diffAssgn: $\forall s. P\ s \longrightarrow (\forall t \geq 0. G\ (sol\ s[xfList \leftarrow uInput]\ t) \longrightarrow Q\ (sol\ s[xfList \leftarrow uInput]\ t))$ 
shows PRE  $P$  (ODEsystem  $xfList$  with  $G$ ) POST  $Q$ 
apply(clarsimp, subgoal-tac  $a=b$ )
apply(clarify, subst boxProgrPred-chrcrtn)
apply(simp-all add: p2r-def)
apply(rule-tac  $uInput=uInput$  in prelim-dSolve)
apply(simp add: solHyp, simp add: uniqHyp)
by (metis (no-types, lifting) diffAssgn)

```

— We proceed to refine the previous rule by finding the necessary restrictions on varFunList and uInput so that the solution to the store-IVP is guaranteed.

**lemma** conds4vdiffs-prelim:

```

assumes funcsHyp: $\forall s\ g. \forall xf \in set\ xfList. \pi_2\ xf\ (override-on\ s\ g\ varDiffs) = \pi_2\ xf\ s$ 
  and distinctHyp:distinct (map  $\pi_1\ xfList$ )
  and varsHyp: $\forall xf \in set\ xfList. \pi_1\ xf \notin varDiffs$ 
  and lengthHyp:length  $xfList = length\ uInput$ 
  and solHyp1: $\forall uxf \in set\ (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$ 
  and solHyp2: $\forall t \geq 0. ((\lambda \tau. (sol\ s[xfList \leftarrow uInput]\ \tau)\ x)$ 
has-vderiv-on  $(\lambda \tau. f\ (sol\ s[xfList \leftarrow uInput]\ \tau)))\ \{0..t\}$ 
  and xfHyp: $(x, f) \in set\ xfList$  and tHyp: $t \geq 0$ 
shows  $(sol\ s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = f\ (sol\ s[xfList \leftarrow uInput]\ t)$ 

```

**proof**—

```

from xfHyp obtain  $u$  where  $xfuHyp: (u, x, f) \in set\ (uInput \otimes xfList)$ 
  by (metis in-set-impl-in-set-zip2 lengthHyp)
show  $(sol\ s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = f\ (sol\ s[xfList \leftarrow uInput]\ t)$ 
proof(cases  $t=0$ )
  case True
  have  $(sol\ s[xfList \leftarrow uInput]\ 0)\ (\partial\ x) = f\ (sol\ s[xfList \leftarrow uInput]\ 0)$ 
    using assms and to-sol-zero-its-dvars by blast
  then show ?thesis using True by blast

```

**next**

```

  case False
  from this have  $t > 0$  using tHyp by simp
  hence  $(sol\ s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = vderiv-of\ (\lambda r. u\ r\ (sol\ s))\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t$ 
    using xfuHyp assms to-sol-greater-than-zero-its-dvars by blast
  also have  $vderiv-of\ (\lambda r. u\ r\ (sol\ s))\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t = f\ (sol\ s[xfList \leftarrow uInput]\ t)$ 
    using assms xfuHyp  $\langle t > 0 \rangle$  and vderiv-of-to-sol-its-vars by blast
  ultimately show ?thesis by simp

```

**qed**

**qed**

**lemma** conds4vdiffs:

```

assumes funcsHyp: $\forall s\ g. \forall xf \in set\ xfList. \pi_2\ xf\ (override-on\ s\ g\ varDiffs) = \pi_2\ xf\ s$ 
  and distinctHyp:distinct (map  $\pi_1\ xfList$ )
  and varsHyp: $\forall xf \in set\ xfList. \pi_1\ xf \notin varDiffs$ 
  and lengthHyp:length  $xfList = length\ uInput$ 
  and solHyp1: $\forall uxf \in set\ (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$ 
  and solHyp2: $\forall t \geq 0. \forall xf \in set\ xfList. ((\lambda \tau. (sol\ s[xfList \leftarrow uInput]\ \tau)\ (\pi_1\ xf))$ 
has-vderiv-on  $(\lambda \tau. (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau)))\ \{0..t\}$ 

```

**shows**  $\forall t \geq 0. \forall xf \in \text{set } xfList. (\text{sol } s[xfList \leftarrow uInput] \ t) \ (\partial \ (\pi_1 \ xf)) =$   
 $(\pi_2 \ xf) \ (\text{sol } s[xfList \leftarrow uInput] \ t)$   
**apply**(rule allI, rule impI, rule ballI, rule conds4vdiffs-prelim)  
**using** *assms* **by** *simp-all*

**lemma** *conds4Consts*:

**assumes** *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$   
**shows**  $\forall x. x \notin (\pi_1 \ (\text{set } xfList)) \longrightarrow (\text{sol } s[xfList \leftarrow uInput] \ t) \ (\partial \ x) = 0$   
**using** *varsHyp* **apply**(induct *xfList* *uInput* rule: *list-induct2*)  
**apply**(*simp-all* add: *override-on-def* *varDiffs-def* *vdifff-def*)  
**by** *clarsimp*

**lemma** *conds4InitState*:

**assumes** *distinctHyp*:*distinct* (map  $\pi_1$  *xfList*)  
**and** *lengthHyp*:*length* *xfList* = *length* *uInput*  
**and** *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$   
**and** *solHyp1*: $\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1 \ uxf) \ 0 \ (\text{sol } s) = (\text{sol } s) \ (\pi_1 \ (\pi_2 \ uxf))$   
**and** *xfHyp*: $(x, f) \in \text{set } xfList$   
**shows**  $(\text{sol } s[xfList \leftarrow uInput] \ 0) \ x = s \ x$

**proof**—

**from** *xfHyp* **obtain** *u* **where** *uxfHyp*: $(u, x, f) \in \text{set } (uInput \otimes xfList)$   
**by** (*metis in-set-impl-in-set-zip2 lengthHyp*)  
**from** *varsHyp* **have** *toZeroHyp*: $(\text{sol } s) \ x = s \ x$  **using** *override-on-def* *xfHyp* **by** *auto*  
**from** *uxfHyp* **and** *solHyp1* **have** *u*  $0 \ (\text{sol } s) = (\text{sol } s) \ x$  **by** *fastforce*  
**also have**  $(\text{sol } s[xfList \leftarrow uInput] \ 0) \ x = u \ 0 \ (\text{sol } s)$   
**using** *state-list-cross-upd-its-vars* *uxfHyp* **and** *assms* **by** *blast*  
**ultimately show**  $(\text{sol } s[xfList \leftarrow uInput] \ 0) \ x = s \ x$  **using** *toZeroHyp* **by** *simp*

**qed**

**lemma** *conds4RestOfStrings*:

**assumes**  $x \notin (\pi_1 \ (\text{set } xfList)) \cup \text{varDiffs}$   
**shows**  $(\text{sol } s[xfList \leftarrow uInput] \ t) \ x = s \ x$   
**using** *assms* **apply**(induct *xfList* *uInput* rule: *list-induct2*)  
**by**(*auto simp: varDiffs-def*)

**lemma** *conds4storeIVP-on-toSol*:

**assumes** *funcsHyp*: $\forall s \ g. \forall xf \in \text{set } xfList. \pi_2 \ xf \ (\text{override-on } s \ g \ \text{varDiffs}) = \pi_2 \ xf \ s$   
**and** *distinctHyp*:*distinct* (map  $\pi_1$  *xfList*)  
**and** *lengthHyp*:*length* *xfList* = *length* *uInput*  
**and** *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$   
**and** *solHyp1*: $\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1 \ uxf) \ 0 \ (\text{sol } s) = (\text{sol } s) \ (\pi_1 \ (\pi_2 \ uxf))$   
**and** *solHyp2*: $\forall t \geq 0. \forall xf \in \text{set } xfList.$   
 $((\lambda t. (\text{sol } s[xfList \leftarrow uInput] \ t) \ (\pi_1 \ xf)) \text{ has-vderiv-on } (\lambda t. \pi_2 \ xf \ (\text{sol } s[xfList \leftarrow uInput] \ t))) \ \{0..t\}$   
**shows** *solvesStoreIVP*  $(\lambda t. (\text{sol } s[xfList \leftarrow uInput] \ t)) \ xfList \ s$   
**apply**(rule *solves-store-ivpI*)  
**subgoal using** *conds4vdiffs* *assms* **by** *blast*  
**subgoal using** *conds4RestOfStrings* **by** *blast*  
**subgoal using** *conds4Consts* *varsHyp* **by** *blast*  
**subgoal apply**(rule allI, rule impI, rule ballI, rule *solves-odeI*)  
**using** *solHyp2* **by** *simp-all*  
**subgoal using** *conds4InitState* **and** *assms* **by** *force*  
**done**

**theorem** *dSolve-toSolve*:

**assumes** *funcsHyp*: $\forall s \ g. \forall xf \in \text{set } xfList. \pi_2 \ xf \ (\text{override-on } s \ g \ \text{varDiffs}) = \pi_2 \ xf \ s$   
**and** *distinctHyp*:*distinct* (map  $\pi_1$  *xfList*)  
**and** *lengthHyp*:*length* *xfList* = *length* *uInput*  
**and** *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$   
**and** *solHyp1*: $\forall s. \forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1 \ uxf) \ 0 \ (\text{sol } s) = (\text{sol } s) \ (\pi_1 \ (\pi_2 \ uxf))$   
**and** *solHyp2*: $\forall s. \forall t \geq 0. \forall xf \in \text{set } xfList.$

$((\lambda t. (sol\ s[xfList \leftarrow uInput]\ t)\ (\pi_1\ xf))\ has\_vderiv\_on\ (\lambda t. \pi_2\ xf\ (sol\ s[xfList \leftarrow uInput]\ t)))\ \{0..t\}$   
**and**  $uniqHyp: \forall s. \forall X. solvesStoreIVP\ X\ xfList\ s \longrightarrow (\forall t \geq 0. (sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$   
**and**  $postCondHyp: \forall s. P\ s \longrightarrow (\forall t \geq 0. Q\ (sol\ s[xfList \leftarrow uInput]\ t))$   
**shows**  $PRE\ P\ (ODEsystem\ xfList\ with\ G)\ POST\ Q$   
**apply**  $(rule\_tac\ uInput = uInput\ in\ dSolve)$   
**subgoal using**  $assms\ and\ conds4storeIVP\_on\_toSol\ by\ simp$   
**subgoal by**  $(simp\ add: uniqHyp)$   
**using**  $postCondHyp\ postCondHyp\ by\ simp$

— As before, we keep refining the rule dSolve. This time we find the necessary restrictions to attain uniqueness.

**lemma**  $conds4UniqSol:$

**fixes**  $f::real\ store \Rightarrow real$   
**assumes**  $tHyp:t \geq 0$   
**and**  $contHyp:continuous\_on\ (\{0..t\} \times UNIV)\ (\lambda(t, (r::real)). f\ (\varphi_s\ t))$   
**shows**  $unique\_on\_bounded\_closed\ 0\ \{0..t\}\ \tau\ (\lambda t\ r. f\ (\varphi_s\ t))\ UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$   
**apply**  $(simp\ add: ubc\_definitions,\ rule\ conjI)$   
**subgoal using**  $contHyp\ continuous\_rhs\_def\ by\ fastforce$   
**subgoal using**  $assms\ continuous\_rhs\_def\ by\ fastforce$   
**done**

**lemma**  $solves\_store\_ivp\_at\_beginning\_overrides:$

**assumes**  $solvesStoreIVP\ \varphi_s\ xfList\ a$   
**shows**  $\varphi_s\ 0 = override\_on\ a\ (\varphi_s\ 0)\ varDiffs$   
**apply**  $(rule\ ext,\ subgoal\_tac\ x \notin varDiffs \longrightarrow \varphi_s\ 0\ x = a\ x)$   
**subgoal by**  $(simp\ add: override\_on\_def)$   
**using**  $assms\ and\ solves\_store\_ivpD(6)\ by\ simp$

**lemma**  $ubcStoreUniqueSol:$

**assumes**  $tHyp:t \geq 0$   
**assumes**  $contHyp:\forall xf \in set\ xfList. continuous\_on\ (\{0..t\} \times UNIV)$   
 $(\lambda(t, (r::real)). (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ t))$   
**and**  $eqDerivs:\forall xf \in set\ xfList. \forall \tau \in \{0..t\}. (\pi_2\ xf)\ (\varphi_s\ \tau) = (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau)$   
**and**  $Fsolves:solvesStoreIVP\ \varphi_s\ xfList\ s$   
**and**  $solHyp:solvesStoreIVP\ (\lambda \tau. (sol\ s[xfList \leftarrow uInput]\ \tau))\ xfList\ s$   
**shows**  $(sol\ s[xfList \leftarrow uInput]\ t) = \varphi_s\ t$

**proof**

**fix**  $x::string$  **show**  $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x$   
**proof**  $(cases\ x \in (\pi_1(\setset xfList)) \cup varDiffs)$   
**case**  $False$   
**then have**  $notInVars:x \notin (\pi_1(\setset xfList)) \cup varDiffs\ by\ simp$   
**from**  $solHyp$  **have**  $(sol\ s[xfList \leftarrow uInput]\ t)\ x = s\ x$   
**using**  $tHyp\ notInVars\ solves\_store\_ivpD(1)\ by\ blast$   
**also from**  $Fsolves$  **have**  $\varphi_s\ t\ x = s\ x\ using\ tHyp\ notInVars\ solves\_store\_ivpD(1)\ by\ blast$   
**ultimately show**  $(sol\ s[xfList \leftarrow uInput]\ t)\ x = \varphi_s\ t\ x\ by\ simp$

**next case**  $True$

**then have**  $x \in (\pi_1(\setset xfList)) \vee x \in varDiffs\ by\ simp$

**from this show**  $?thesis$

**proof**

**assume**  $x \in (\pi_1(\setset xfList))$

**from this obtain**  $f$  **where**  $xfHyp:(x, f) \in set\ xfList\ by\ fastforce$

**then have**  $expand1:\forall xf \in set\ xfList. ((\lambda \tau. \varphi_s\ \tau\ (\pi_1\ xf))\ solves\_ode$

$(\lambda \tau\ r. (\pi_2\ xf)\ (\varphi_s\ \tau)))\ \{0..t\}\ UNIV \wedge \varphi_s\ 0\ (\pi_1\ xf) = s\ (\pi_1\ xf)$

**using**  $Fsolves\ tHyp\ by\ (simp\ add:solvesStoreIVP\_def)$

**hence**  $expand2:\forall xf \in set\ xfList. \forall \tau \in \{0..t\}. ((\lambda r. \varphi_s\ r\ (\pi_1\ xf))$

$has\_vector\_derivative\ (\lambda r. (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau))\ \tau)\ (at\ \tau\ within\ \{0..t\})$

**using**  $eqDerivs\ by\ (simp\ add: solves\_ode\_def\ has\_vderiv\_on\_def)$

**then have**  $\forall xf \in \text{set } xfList. ((\lambda \tau. \varphi_s \tau (\pi_1 xf)) \text{ solves-ode } (\lambda \tau r. (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] \tau))) \{0..t\} UNIV \wedge \varphi_s\ 0\ (\pi_1 xf) = s\ (\pi_1 xf)$   
**by** (simp add: has-vderiv-on-def solves-ode-def expand1 expand2)  
**then have**  $1:((\lambda \tau. \varphi_s \tau x) \text{ solves-ode } (\lambda \tau r. f\ (sol\ s[xfList \leftarrow uInput] \tau))) \{0..t\} UNIV \wedge \varphi_s\ 0\ x = s\ x$  **using** *xfHyp* **by** *fastforce*

**from** *solHyp* **and** *xfHyp* **have**  $2:((\lambda \tau. (sol\ s[xfList \leftarrow uInput] \tau)\ x) \text{ solves-ode } (\lambda \tau r. f\ (sol\ s[xfList \leftarrow uInput] \tau))) \{0..t\} UNIV \wedge (sol\ s[xfList \leftarrow uInput] 0)\ x = s\ x$   
**using** *solvesStoreIVP-def tHyp* **by** *fastforce*

**from** *tHyp* **and** *contHyp* **have**  $\forall xf \in \text{set } xfList. \text{unique-on-bounded-closed } 0\ \{0..t\}\ (s\ (\pi_1 xf))\ (\lambda \tau r. (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] \tau))\ UNIV\ (\text{if } t = 0 \text{ then } 1 \text{ else } 1/(t+1))$   
**apply**(clarify) **apply**(rule *conds4UniqSol*) **by** *auto*  
**from this** **have**  $3:\text{unique-on-bounded-closed } 0\ \{0..t\}\ (s\ x)\ (\lambda \tau r. f\ (sol\ s[xfList \leftarrow uInput] \tau))\ UNIV\ (\text{if } t = 0 \text{ then } 1 \text{ else } 1/(t+1))$  **using** *xfHyp* **by** *fastforce*  
**from** 1 2 **and** 3 **show**  $(sol\ s[xfList \leftarrow uInput] t)\ x = \varphi_s\ t\ x$   
**using** *unique-on-bounded-closed.unique-solution* **using** *real-Icc-closed-segment tHyp* **by** *blast*

**next**  
**assume**  $x \in \text{varDiffs}$   
**then obtain** *y* **where**  $xDef: x = \partial\ y$  **by** (*auto simp: varDiffs-def*)  
**show**  $(sol\ s[xfList \leftarrow uInput] t)\ x = \varphi_s\ t\ x$   
**proof**(cases  $y \in \text{set } (\text{map } \pi_1\ xfList)$ )  
**case** *True*  
**then obtain** *f* **where**  $xfHyp:(y, f) \in \text{set } xfList$  **by** *fastforce*  
**from** *tHyp* **and** *Fsolves* **have**  $\varphi_s\ t\ x = f\ (\varphi_s\ t)$   
**using** *solves-store-ivpD(3) xfHyp xDef* **by** *force*  
**also have**  $(sol\ s[xfList \leftarrow uInput] t)\ x = f\ (sol\ s[xfList \leftarrow uInput] t)$   
**using** *solves-store-ivpD(3) xfHyp xDef solHyp tHyp* **by** *force*  
**ultimately show** *?thesis* **using** *eqDerivs xfHyp tHyp* **by** *auto*

**next case** *False*  
**then have**  $\varphi_s\ t\ x = 0$   
**using** *xDef solves-store-ivpD(2) Fsolves tHyp* **by** *simp*  
**also have**  $(sol\ s[xfList \leftarrow uInput] t)\ x = 0$   
**using** *False solHyp tHyp solves-store-ivpD(2) xDef* **by** *fastforce*  
**ultimately show** *?thesis* **by** *simp*

**qed**  
**qed**  
**qed**  
**qed**

**theorem** *dSolveUBC*:  
**assumes** *contHyp*: $\forall s. \forall t \geq 0. \forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$   
 $(\lambda(t, (r::real)). (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] t))$   
**and** *solHyp*: $\forall s. \text{solvesStoreIVP } (\lambda t. (sol\ s[xfList \leftarrow uInput] t))\ xfList\ s$   
**and** *uniqHyp*: $\forall s. \forall \varphi_s. \varphi_s \text{ solvesTheStoreIVP } xfList\ \text{withInitState } s \longrightarrow$   
 $(\forall t \geq 0. \forall xf \in \text{set } xfList. \forall r \in \{0..t\}. (\pi_2 xf) (\varphi_s\ r) = (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] r))$   
**and** *diffAssgn*: $\forall s. P\ s \longrightarrow (\forall t \geq 0. G\ (sol\ s[xfList \leftarrow uInput] t) \longrightarrow Q\ (sol\ s[xfList \leftarrow uInput] t))$   
**shows** *PRE* *P* (*ODEsystem* *xfList* with *G*) *POST* *Q*  
**apply**(rule-tac *uInput=uInput* **in** *dSolve*)  
**prefer** 2 **subgoal** **proof**(clarify)  
**fix** *s::real store* **and**  $\varphi_s::real \Rightarrow \text{real store}$  **and** *t::real*  
**assume** *isSol:solvesStoreIVP*  $\varphi_s\ xfList\ s$  **and** *sHyp*: $0 \leq t$   
**from this** **and** *uniqHyp* **have**  $\forall xf \in \text{set } xfList. \forall t \in \{0..t\}. (\pi_2 xf) (\varphi_s\ t) = (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] t)$  **by** *auto*  
**also have**  $\forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$   
 $(\lambda(t, (r::real)). (\pi_2 xf) (sol\ s[xfList \leftarrow uInput] t))$  **using** *contHyp sHyp* **by** *blast*  
**ultimately show**  $(sol\ s[xfList \leftarrow uInput] t) = \varphi_s\ t$   
**using** *sHyp isSol ubcStoreUniqueSol solHyp* **by** *simp*  
**qed** **using** *assms* **by** *simp-all*

**theorem** *dSolve-toSolveUBC*:

**assumes** *funcsHyp*: $\forall s\ g.\ \forall xf \in \text{set } xfList.\ \pi_2\ xf\ (\text{override-on } s\ g\ \text{varDiffs}) = \pi_2\ xf\ s$   
**and** *distinctHyp*: $\text{distinct } (\text{map } \pi_1\ xfList)$   
**and** *lengthHyp*: $\text{length } xfList = \text{length } uInput$   
**and** *varsHyp*: $\forall xf \in \text{set } xfList.\ \pi_1\ xf \notin \text{varDiffs}$   
**and** *solHyp1*: $\forall s.\ \forall uxf \in \text{set } (uInput \otimes xfList).\ \pi_1\ uxf\ 0\ (\text{sol } s) = \text{sol } s\ (\pi_1\ (\pi_2\ uxf))$   
**and** *solHyp2*: $\forall s.\ \forall t \geq 0.\ \forall xf \in \text{set } xfList.\ ((\lambda t.\ (\text{sol } s[xfList \leftarrow uInput]\ t)\ (\pi_1\ xf)))\ \text{has-vderiv-on}$   
 $(\lambda t.\ \pi_2\ xf\ (\text{sol } s[xfList \leftarrow uInput]\ t)))\ \{0..t\}$   
**and** *contHyp*: $\forall s.\ \forall t \geq 0.\ \forall xf \in \text{set } xfList.\ \text{continuous-on } (\{0..t\} \times UNIV)$   
 $(\lambda(t, (r::\text{real})).\ (\pi_2\ xf)\ (\text{sol } s[xfList \leftarrow uInput]\ t))$   
**and** *uniqHyp*: $\forall s.\ \forall \varphi_s.\ \varphi_s\ \text{solvesTheStoreIVP } xfList\ \text{withInitState } s \longrightarrow$   
 $(\forall t \geq 0.\ \forall xf \in \text{set } xfList.\ \forall r \in \{0..t\}.\ (\pi_2\ xf)\ (\varphi_s\ r) = (\pi_2\ xf)\ (\text{sol } s[xfList \leftarrow uInput]\ r))$   
**and** *postCondHyp*: $\forall s.\ P\ s \longrightarrow (\forall t \geq 0.\ Q\ (\text{sol } s[xfList \leftarrow uInput]\ t))$   
**shows** *PRE* *P* (*ODEsystem* *xfList* with *G*) *POST* *Q*  
**apply**(*rule-tac* *uInput=uInput* **in** *dSolveUBC*)  
**using** *contHyp* **apply** *simp*  
**apply**(*rule allI*, *rule-tac* *uInput=uInput* **in** *conds4storeIVP-on-toSol*)  
**using** *assms* **by** *auto*

**”Differential Invariant.”**

**lemma** *solvesStoreIVP-couldBeModified*:

**fixes** *F*: $\text{real} \Rightarrow \text{real store}$   
**assumes** *vars*: $\forall t \geq 0.\ \forall xf \in \text{set } xfList.\ ((\lambda t.\ F\ t\ (\pi_1\ xf)))\ \text{solves-ode } (\lambda t\ r.\ \pi_2\ xf\ (F\ t)))\ \{0..t\}\ UNIV$   
**and** *dvars*: $\forall t \geq 0.\ \forall xf \in \text{set } xfList.\ (F\ t\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (F\ t)$   
**shows**  $\forall t \geq 0.\ \forall r \in \{0..t\}.\ \forall xf \in \text{set } xfList.$   
 $((\lambda t.\ F\ t\ (\pi_1\ xf)))\ \text{has-vector-derivative } F\ r\ (\partial\ (\pi_1\ xf))\ (\text{at } r\ \text{within } \{0..t\})$   
**proof**(*clarify*, *rename-tac* *t r x f*)  
**fix** *x f* **and** *t r*: $\text{real}$   
**assume** *tHyp*: $0 \leq t$  **and** *xfHyp*: $(x, f) \in \text{set } xfList$  **and** *rHyp*: $r \in \{0..t\}$   
**from this and vars have**  $((\lambda t.\ F\ t\ x)\ \text{solves-ode } (\lambda t\ r.\ f\ (F\ t)))\ \{0..t\}\ UNIV$   
**using** *tHyp* **by** *fastforce*  
**hence**  $\forall r \in \{0..t\}.\ ((\lambda t.\ F\ t\ x)\ \text{has-vector-derivative } (\lambda t.\ f\ (F\ t))\ r)\ (\text{at } r\ \text{within } \{0..t\})$   
**by** (*simp add: solves-ode-def has-vderiv-on-def tHyp*)  
**have**  $\forall t \geq 0.\ \forall r \in \{0..t\}.\ \forall xf \in \text{set } xfList.\ (F\ r\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (F\ r)$   
**using** *assms* **by** *auto*  
**from this rHyp and xfHyp have**  $(F\ r\ (\partial\ x)) = f\ (F\ r)$   
**by** *force*  
**then show**  $((\lambda t.\ F\ t\ (\pi_1\ (x, f)))\ \text{has-vector-derivative } F\ r\ (\partial\ (\pi_1\ (x, f))))\ (\text{at } r\ \text{within } \{0..t\})$   
**using**  $\ast\ rHyp$  **by** *auto*  
**qed**

**lemma** *derivationLemma-baseCase*:

**fixes** *F*: $\text{real} \Rightarrow \text{real store}$   
**assumes** *solves*:*solvesStoreIVP* *F* *xfList* *a*  
**shows**  $\forall x \in (UNIV - \text{varDiffs}).\ \forall t \geq 0.\ \forall r \in \{0..t\}.$   
 $((\lambda t.\ F\ t\ x)\ \text{has-vector-derivative } F\ r\ (\partial\ x))\ (\text{at } r\ \text{within } \{0..t\})$   
**proof**  
**fix** *x*  
**assume**  $x \in UNIV - \text{varDiffs}$   
**then have** *notVarDiff*: $\forall z.\ x \neq \partial\ z$  **using** *varDiffs-def* **by** *fastforce*  
**show**  $\forall t \geq 0.\ \forall r \in \{0..t\}.\ ((\lambda t.\ F\ t\ x)\ \text{has-vector-derivative } F\ r\ (\partial\ x))\ (\text{at } r\ \text{within } \{0..t\})$   
**proof**(*cases*  $x \in \text{set } (\text{map } \pi_1\ xfList)$ )  
**case** *True*  
**from this and solves have**  $\forall t \geq 0.\ \forall r \in \{0..t\}.\ \forall xf \in \text{set } xfList.$   
 $((\lambda t.\ F\ t\ (\pi_1\ xf)))\ \text{has-vector-derivative } F\ r\ (\partial\ (\pi_1\ xf))\ (\text{at } r\ \text{within } \{0..t\})$   
**apply**(*rule-tac* *solvesStoreIVP-couldBeModified*) **using** *solves solves-store-ivpD* **by** *auto*  
**from this show** *?thesis* **using** *True* **by** *auto*  
**next**  
**case** *False*



from this notVarDiff and solves have const: $\forall t \geq 0. F t x = a x$   
 using solves-store-ivpD(1) by (simp add: varDiffs-def)  
 have constD: $\forall t \geq 0. \forall r \in \{0..t\}. ((\lambda r. a x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$   
 by (auto intro: derivative-eq-intros)  
 {fix t r::real  
 assume  $t \geq 0$  and  $r \in \{0..t\}$   
 hence  $((\lambda s. a x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$  by (simp add: constD)  
 moreover have  $\bigwedge s. s \in \{0..t\} \implies (\lambda r. F r x) s = (\lambda r. a x) s$   
 using const by (simp add:  $\langle 0 \leq t \rangle$ )  
 ultimately have  $((\lambda s. F s x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$   
 using has-vector-derivative-transform by (metis  $\langle r \in \{0..t\} \rangle$ )  
 hence isZero: $\forall t \geq 0. \forall r \in \{0..t\}. ((\lambda t. F t x) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$   
 by blast  
 from False solves and notVarDiff have  $\forall t \geq 0. F t (\partial x) = 0$   
 using solves-store-ivpD(2) by simp  
 then show ?thesis using isZero by simp  
 qed  
 qed

lemma derivationLemma:

assumes solvesStoreIVP F xflist a  
 and tHyp: $t \geq 0$   
 and termVarsHyp: $\forall x \in \text{trmVars } \eta. x \in (\text{UNIV} - \text{varDiffs})$   
 shows  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F r)) \text{ (at } r \text{ within } \{0..t\})$   
 using termVarsHyp proof(induction  $\eta$ )  
 case (Const r)  
 then show ?case by simp  
 next  
 case (Var y)  
 then have yHyp: $y \in \text{UNIV} - \text{varDiffs}$  by auto  
 from this tHyp and assms(1) show ?case  
 using derivationLemma-baseCase by auto  
 next  
 case (Mns  $\eta$ )  
 then show ?case  
 apply(clarsimp)  
 by(rule derivative-intros, simp)  
 next  
 case (Sum  $\eta1 \ \eta2$ )  
 then show ?case  
 apply(clarsimp)  
 by(rule derivative-intros, simp-all)  
 next  
 case (Mult  $\eta1 \ \eta2$ )  
 then show ?case  
 apply(clarsimp)  
 apply(subgoal-tac  $((\lambda s. \llbracket \eta1 \rrbracket_t (F s) *_R \llbracket \eta2 \rrbracket_t (F s)) \text{ has-vector-derivative } \llbracket \partial_t \eta1 \rrbracket_t (F r) \cdot \llbracket \eta2 \rrbracket_t (F r) + \llbracket \eta1 \rrbracket_t (F r) \cdot \llbracket \partial_t \eta2 \rrbracket_t (F r)) \text{ (at } r \text{ within } \{0..t\}), \text{simp}$ )  
 apply(rule-tac  $f'1 = \llbracket \partial_t \eta1 \rrbracket_t (F r)$  and  $g'1 = \llbracket \partial_t \eta2 \rrbracket_t (F r)$  in derivative-eq-intros(26))  
 by (simp-all add: has-field-derivative-iff-has-vector-derivative)  
 qed

lemma diff-subst-prprty-4terms:

assumes solves: $\forall xf \in \text{set } xflist. F t (\partial (\pi_1 xf)) = \pi_2 xf (F t)$   
 and tHyp: $(t::\text{real}) \geq 0$   
 and listsHyp: $\text{map } \pi_2 xflist = \text{map } \text{tval } uInput$   
 and termVarsHyp: $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$   
 shows  $\llbracket \partial_t \eta \rrbracket_t (F t) = \llbracket ((\text{map } (\text{vdiff} \circ \pi_1) xflist) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F t)$   
 using termVarsHyp apply(induction  $\eta$ ) apply(simp-all add: substList-help2)  
 using listsHyp and solves apply(induct xflist uInput rule: list-induct2', simp, simp, simp)

**proof**(clarify, rename-tac  $y \ g \ xfTail \ \vartheta \ trmTail \ x$ )  
**fix**  $x \ y :: string$  **and**  $\vartheta :: trms$  **and**  $g$  **and**  $xfTail :: (string \times (real \ store \Rightarrow real)) \ list$  **and**  $trmTail$   
**assume**  $IH : \bigwedge x. x \notin varDiffs \Rightarrow map \ \pi_2 \ xfTail = map \ tval \ trmTail \Rightarrow$   
 $\forall xf \in set \ xfTail. F \ t \ (\partial \ (\pi_1 \ xf)) = \pi_2 \ xf \ (F \ t) \Rightarrow$   
 $F \ t \ (\partial \ x) = \llbracket (map \ (vdiff \circ \pi_1) \ xfTail \otimes trmTail) \langle t_V \ (\partial \ x) \rangle \rrbracket_t (F \ t)$   
**and**  $1 : x \notin varDiffs$  **and**  $2 : map \ \pi_2 \ ((y, g) \# xfTail) = map \ tval \ (\vartheta \# trmTail)$   
**and**  $3 : \forall xf \in set \ ((y, g) \# xfTail). F \ t \ (\partial \ (\pi_1 \ xf)) = \pi_2 \ xf \ (F \ t)$   
**hence**  $*$  :  $\llbracket (map \ (vdiff \circ \pi_1) \ xfTail \otimes trmTail) \langle Var \ (\partial \ x) \rangle \rrbracket_t (F \ t) = F \ t \ (\partial \ x)$   
**using**  $tHyp$  **by**  $auto$   
**show**  $F \ t \ (\partial \ x) = \llbracket ((map \ (vdiff \circ \pi_1) \ ((y, g) \# xfTail)) \otimes (\vartheta \# trmTail)) \langle t_V \ (\partial \ x) \rangle \rrbracket_t (F \ t)$   
**proof**(cases  $x \in set \ (map \ \pi_1 \ ((y, g) \# xfTail))$ )  
**case**  $True$   
**then** have  $x = y \vee (x \neq y \wedge x \in set \ (map \ \pi_1 \ xfTail))$  **by**  $auto$   
**moreover**  
**{** **assume**  $x = y$   
**from**  $this$  **have**  $((map \ (vdiff \circ \pi_1) \ ((y, g) \# xfTail)) \otimes (\vartheta \# trmTail)) \langle t_V \ (\partial \ x) \rangle = \vartheta$   
**by**  $simp$   
**also** **from**  $3 \ tHyp$  **have**  $F \ t \ (\partial \ y) = g \ (F \ t)$   
**by**  $simp$   
**moreover** **from**  $2$  **have**  $\llbracket \vartheta \rrbracket_t (F \ t) = g \ (F \ t)$   
**by**  $simp$   
**ultimately** **have**  $?thesis$   
**by**  $(simp \ add : \langle x = y \rangle)$   
**moreover**  
**{** **assume**  $x \neq y \wedge x \in set \ (map \ \pi_1 \ xfTail)$   
**then** **have**  $\partial \ x \neq \partial \ y$  **using**  $vdiff\text{-}inj$  **by**  $auto$   
**from**  $this$  **have**  $((map \ (vdiff \circ \pi_1) \ ((y, g) \# xfTail)) \otimes (\vartheta \# trmTail)) \langle t_V \ (\partial \ x) \rangle =$   
 $((map \ (vdiff \circ \pi_1) \ xfTail) \otimes trmTail) \langle t_V \ (\partial \ x) \rangle$  **by**  $simp$   
**hence**  $?thesis$  **using**  $*$  **by**  $simp$   
**ultimately** **show**  $?thesis$  **by**  $blast$   
**next**  
**case**  $False$   
**then** **have**  $((map \ (vdiff \circ \pi_1) \ ((y, g) \# xfTail)) \otimes (\vartheta \# trmTail)) \langle t_V \ (\partial \ x) \rangle = t_V \ (\partial \ x)$   
**using**  $substList\text{-}cross\text{-}vdiff\text{-}on\text{-}non\text{-}occurring\text{-}var$   
**by**  $(metis(no\text{-}types, lifting) \ List.map.compositionality)$   
**thus**  $?thesis$  **by**  $simp$   
**qed**  
**qed**

**lemma**  $eqInVars\text{-}impl\text{-}eqInTrms$ :

**assumes**  $termVarsHyp : trmVars \ \eta \subseteq (UNIV - varDiffs)$   
**and**  $initHyp : \forall x. x \notin varDiffs \longrightarrow b \ x = a \ x$   
**shows**  $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t b$   
**using**  $assms$  **by**  $(induction \ \eta, simp\text{-}all)$

**lemma**  $non\text{-}empty\text{-}funList\text{-}implies\text{-}non\text{-}empty\text{-}trmList$ :

**shows**  $\forall list. (x, f) \in set \ list \wedge map \ \pi_2 \ list = map \ tval \ tList \longrightarrow (\exists \vartheta. \llbracket \vartheta \rrbracket_t = f \wedge \vartheta \in set \ tList)$   
**by**  $(induction \ tList, auto)$

**lemma**  $dInvForTrms\text{-}prelim$ :

**assumes**  $substHyp$ :  
 $\forall st. G \ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket set \ xfList \rrbracket)) \longrightarrow st \ (\partial \ str) = 0 \longrightarrow$   
 $\llbracket ((map \ (vdiff \circ \pi_1) \ xfList) \otimes uInput) \langle \partial_t \ \eta \rangle \rrbracket_t st = 0$   
**and**  $termVarsHyp : trmVars \ \eta \subseteq (UNIV - varDiffs)$   
**and**  $listsHyp : map \ \pi_2 \ xfList = map \ tval \ uInput$   
**shows**  $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem \ xfList \text{ with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0)$

**proof**(clarify)

**fix**  $c$  **assume**  $aHyp : \llbracket \eta \rrbracket_t a = 0$  **and**  $cHyp : (a, c) \in ODEsystem \ xfList \text{ with } G$   
**from**  $this$  **obtain**  $t :: real$  **and**  $F :: real \Rightarrow real \ store$   
**where**  $tcHyp : t \geq 0 \wedge F \ t = c \wedge solvesStoreIVP \ F \ xfList \ a \wedge (\forall r \in \{0..t\}. G \ (F \ r))$

```

    using guarDiffEqtn-def by auto
  then have  $\forall x. x \notin \text{varDiffs} \longrightarrow F\ 0\ x = a\ x$ 
    using solves-store-ivpD(6) by blast
  from this have  $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t (F\ 0)$ 
    using termVarsHyp eqInVars-impl-eqInTrms by blast
  hence  $\text{obs1}:\llbracket \eta \rrbracket_t (F\ 0) = 0$ 
    using aHyp by simp
  hence  $\text{obs2}:\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F\ r)) \text{ (at } r \text{ within } \{0..t\})$ 
    using tcHyp derivationLemma termVarsHyp by blast
  have  $\forall r \in \{0..t\}. \forall xf \in \text{set } xfList. F\ r\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ r)$ 
    using tcHyp solves-store-ivpD(3) by fastforce
  hence  $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F\ r) = \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F\ r)$ 
    using tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp by fastforce
  also from substHyp have  $\forall r \in \{0..t\}. \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F\ r) = 0$ 
    using solves-store-ivpD(2) tcHyp by fastforce
  ultimately have  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s)) \text{ has-vector-derivative } 0) \text{ (at } r \text{ within } \{0..t\})$ 
    using obs2 by auto
  hence  $\forall s \in \{0..t\}. ((\lambda x. \llbracket \eta \rrbracket_t (F\ x)) \text{ has-derivative } (\lambda x. x *_R 0)) \text{ (at } s \text{ within } \{0..t\})$ 
    using tcHyp by (metis has-vector-derivative-def)
  hence  $\llbracket \eta \rrbracket_t (F\ t) - \llbracket \eta \rrbracket_t (F\ 0) = (\lambda x. x *_R 0) (t - 0)$ 
    using mvt-very-simple and tcHyp by fastforce
  then show  $\llbracket \eta \rrbracket_t c = 0$ 
    using obs1 tcHyp by auto
qed

```

**theorem** *dInvForTrms*:

```

  assumes  $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set } xfList \rrbracket) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$ 
 $\llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t st = 0$ 
    and  $\text{termVarsHyp}:\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$ 
    and  $\text{listsHyp}:\text{map } \pi_2\ xfList = \text{map } \text{tval } uInput$ 
    and  $\text{eta-f}:f = \llbracket \eta \rrbracket_t$ 
  shows  $\text{PRE } (\lambda s. f\ s = 0) \text{ (ODEsystem } xfList \text{ with } G) \text{ POST } (\lambda s. f\ s = 0)$ 
    using eta-f proof(clarsimp)
  fix a b
  assume  $(a, b) \in [\lambda s. \llbracket \eta \rrbracket_t s = 0]$  and  $f = \llbracket \eta \rrbracket_t$ 
  from this have  $aHyp:a = b \wedge \llbracket \eta \rrbracket_t a = 0$ 
    by (simp add: p2r-def)
  have  $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem } xfList \text{ with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0)$ 
    using assms dInvForTrms-prelim by metis
  from this and aHyp have  $\forall c. (a, c) \in (\text{ODEsystem } xfList \text{ with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0$ 
    by blast
  thus  $(a, b) \in wp\ (\text{ODEsystem } xfList \text{ with } G)\ [\lambda s. \llbracket \eta \rrbracket_t s = 0]$ 
    using aHyp by (simp add: boxProgrPred-chrctrztn)
qed

```

**lemma** *diff-subst-prprty-4props*:

```

  assumes solves: $\forall xf \in \text{set } xfList. F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$ 
    and tHyp: $t \geq 0$ 
    and listsHyp: $\text{map } \pi_2\ xfList = \text{map } \text{tval } uInput$ 
    and propVarsHyp: $\text{propVars } \varphi \subseteq (\text{UNIV} - \text{varDiffs})$ 
  shows  $\llbracket \partial_P \varphi \rrbracket_P (F\ t) = \llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \restriction \partial_P \varphi \rrbracket_P (F\ t)$ 
    using propVarsHyp apply(induction  $\varphi$ , simp-all)
    using assms diff-subst-prprty-4terms apply fastforce
    using assms diff-subst-prprty-4terms apply fastforce
    using assms diff-subst-prprty-4terms by fastforce

```

**lemma** *dInvForProps-prelim*:

```

  assumes substHyp:
 $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set } xfList \rrbracket) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$ 
 $\llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t st \geq 0$ 

```

**and**  $\text{termVarsHyp}:\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$   
**and**  $\text{listsHyp}:\text{map } \pi_2 \text{ xflist} = \text{map tval uInput}$   
**shows**  $\llbracket \eta \rrbracket_t a > 0 \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem xflist with } G) \longrightarrow \llbracket \eta \rrbracket_t c > 0)$   
**and**  $\llbracket \eta \rrbracket_t a \geq 0 \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem xflist with } G) \longrightarrow \llbracket \eta \rrbracket_t c \geq 0)$   
**proof**(clarify)  
**fix**  $c$  **assume**  $a\text{Hyp}:\llbracket \eta \rrbracket_t a > 0$  **and**  $c\text{Hyp}:(a, c) \in \text{ODEsystem xflist with } G$   
**from this obtain**  $t::\text{real}$  **and**  $F::\text{real} \Rightarrow \text{real store}$   
**where**  $tc\text{Hyp}:t \geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ xflist } a \wedge (\forall r \in \{0..t\}. G (F r))$   
**using**  $\text{guarDiffEqtn-def}$  **by** *auto*  
**then have**  $\forall x. x \notin \text{varDiffs} \longrightarrow F 0 x = a x$   
**using**  $\text{solves-store-ivpD}(6)$  **by** *blast*  
**from this have**  $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t (F 0)$   
**using**  $\text{termVarsHyp eqInVars-impl-eqInTrms}$  **by** *blast*  
**hence**  $\text{obs1}:\llbracket \eta \rrbracket_t (F 0) > 0$   
**using**  $a\text{Hyp } tc\text{Hyp}$  **by** *simp*  
**from**  $tc\text{Hyp}$  **have**  $\text{obs2}:\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } (\partial_t \llbracket \eta \rrbracket_t (F r))) \text{ (at } r \text{ within } \{0..t\})$   
**using**  $\text{derivationLemma termVarsHyp}$  **by** *blast*  
**have**  $(\forall t \geq 0. \forall x f \in \text{set xflist}. F t (\partial (\pi_1 x f)) = \pi_2 x f (F t))$   
**using**  $tc\text{Hyp solves-store-ivpD}(3)$  **by** *blast*  
**hence**  $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) = \llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xflist}) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F r)$   
**using**  $\text{diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp}$  **by** *fastforce*  
**also from**  $\text{substHyp}$  **have**  $\forall r \in \{0..t\}. \llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xflist}) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F r) \geq 0$   
**using**  $\text{solves-store-ivpD}(2) tc\text{Hyp}$  **by**  $(\text{metis atLeastAtMost-iff})$   
**ultimately have**  $*\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) \geq 0$   
**by** *simp*  
**from**  $\text{obs2}$  **and**  $tc\text{Hyp}$  **have**  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-derivative } (\lambda x. x *_R (\llbracket \partial_t \eta \rrbracket_t (F r)))) \text{ (at } r \text{ within } \{0..t\})$   
**by**  $(\text{simp add: has-vector-derivative-def})$   
**hence**  $\exists r \in \{0..t\}. \llbracket \eta \rrbracket_t (F t) - \llbracket \eta \rrbracket_t (F 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F r))$   
**using**  $\text{mvt-very-simple}$  **and**  $tc\text{Hyp}$  **by** *fastforce*  
**then obtain**  $r$  **where**  $\llbracket \partial_t \eta \rrbracket_t (F r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge \llbracket \partial_t \eta \rrbracket_t (F t) \geq 0$   
 $\wedge \llbracket \eta \rrbracket_t (F t) - \llbracket \eta \rrbracket_t (F 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F r))$   
**using**  $* tc\text{Hyp}$  **by**  $(\text{meson atLeastAtMost-iff order-refl})$   
**thus**  $\llbracket \eta \rrbracket_t c > 0$   
**using**  $\text{obs1 } tc\text{Hyp}$  **by**  $(\text{smt mult-nonneg-nonneg})$   
**next**  
**show**  $0 \leq \llbracket \eta \rrbracket_t a \longrightarrow (\forall c. (a, c) \in \text{ODEsystem xflist with } G \longrightarrow 0 \leq \llbracket \eta \rrbracket_t c)$   
**proof**(clarify)  
**fix**  $c$  **assume**  $a\text{Hyp}:\llbracket \eta \rrbracket_t a \geq 0$  **and**  $c\text{Hyp}:(a, c) \in \text{ODEsystem xflist with } G$   
**from this obtain**  $t::\text{real}$  **and**  $F::\text{real} \Rightarrow \text{real store}$   
**where**  $tc\text{Hyp}:t \geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ xflist } a \wedge (\forall r \in \{0..t\}. G (F r))$   
**using**  $\text{guarDiffEqtn-def}$  **by** *auto*  
**then have**  $\forall x. x \notin \text{varDiffs} \longrightarrow F 0 x = a x$   
**using**  $\text{solves-store-ivpD}(6)$  **by** *blast*  
**from this have**  $\llbracket \eta \rrbracket_t a = \llbracket \eta \rrbracket_t (F 0)$   
**using**  $\text{termVarsHyp eqInVars-impl-eqInTrms}$  **by** *blast*  
**hence**  $\text{obs1}:\llbracket \eta \rrbracket_t (F 0) \geq 0$   
**using**  $a\text{Hyp } tc\text{Hyp}$  **by** *simp*  
**hence**  $\text{obs2}:\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F r)) \text{ (at } r \text{ within } \{0..t\})$   
**using**  $tc\text{Hyp derivationLemma termVarsHyp}$  **by** *blast*  
**have**  $(\forall t \geq 0. \forall x f \in \text{set xflist}. F t (\partial (\pi_1 x f)) = \pi_2 x f (F t))$   
**using**  $tc\text{Hyp solves-store-ivpD}(3)$  **by** *blast*  
**hence**  $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) = \llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xflist}) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F r)$   
**using**  $tc\text{Hyp diff-subst-prprty-4terms termVarsHyp listsHyp}$  **by** *fastforce*  
**also from**  $\text{substHyp}$  **have**  $\forall r \in \{0..t\}. \llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xflist}) \otimes uInput) \langle \partial_t \eta \rangle \rrbracket_t (F r) \geq 0$   
**using**  $\text{solves-store-ivpD}(2) tc\text{Hyp}$  **by**  $(\text{metis atLeastAtMost-iff})$   
**ultimately have**  $*\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) \geq 0$   
**by** *simp*  
**have**  $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-derivative } (\lambda x. x *_R (\llbracket \partial_t \eta \rrbracket_t (F r)))) \text{ (at } r \text{ within } \{0..t\})$

using *obs2 tcHyp* by (*simp add: has-vector-derivative-def*)  
 hence  $\exists r \in \{0..t\}. \llbracket \eta \rrbracket_t (F t) - \llbracket \eta \rrbracket_t (F 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F r))$   
 using *mut-very-simple* and *tcHyp* by *fastforce*  
 then obtain *r* where  $\llbracket \partial_t \eta \rrbracket_t (F r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge \llbracket \partial_t \eta \rrbracket_t (F t) \geq 0$   
 $\wedge \llbracket \eta \rrbracket_t (F t) - \llbracket \eta \rrbracket_t (F 0) = t \cdot (\llbracket \partial_t \eta \rrbracket_t (F r))$   
 using *\* tcHyp* by (*meson atLeastAtMost-iff order-refl*)  
 thus  $\llbracket \eta \rrbracket_t c \geq 0$   
 using *obs1 tcHyp* by (*smt mult-nonneg-nonneg*)  
 qed  
 qed

lemma *less-pval-to-tval*:

assumes  $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_P (\vartheta \prec \eta) \rrbracket_P st$   
 shows  $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_t (\eta \oplus (\ominus \vartheta)) \rrbracket_t st \geq 0$   
 using *assms* by *auto*

lemma *leq-pval-to-tval*:

assumes  $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_P (\vartheta \preceq \eta) \rrbracket_P st$   
 shows  $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_t (\eta \oplus (\ominus \vartheta)) \rrbracket_t st \geq 0$   
 using *assms* by *auto*

lemma *dInv-prelim*:

assumes *substHyp*:  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_P \varphi \rrbracket_P st$   
 and *propVarsHyp*:  $\text{propVars } \varphi \subseteq (UNIV - \text{varDiffs})$   
 and *listsHyp*:  $\text{map } \pi_2 \text{ xfList} = \text{map tval } uInput$   
 shows  $\llbracket \varphi \rrbracket_P a \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem xfList with } G) \longrightarrow \llbracket \varphi \rrbracket_P c)$

proof(*clarify*)

fix *c* assume *aHyp*:  $\llbracket \varphi \rrbracket_P a$  and *cHyp*:  $(a, c) \in \text{ODEsystem xfList with } G$   
 from *this* obtain *t*:*real* and *F*:*real*  $\Rightarrow$  *real* store  
 where *tcHyp*:  $t \geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ xfList } a$   
 using *guarDiffEqtn-def* by *auto*  
 from *aHyp propVarsHyp* and *substHyp* show  $\llbracket \varphi \rrbracket_P c$   
 proof(*induction*  $\varphi$ )  
 case (*Eq*  $\vartheta \eta$ )  
 hence *hyp*:  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_P (\vartheta \doteq \eta) \rrbracket_P st$   
 by *blast*  
 then have  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $\llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xfList}) \otimes uInput) \upharpoonright \partial_t (\vartheta \oplus (\ominus \eta)) \rrbracket_t st = 0$   
 by *simp*  
 also have  $\text{trmVars } (\vartheta \oplus (\ominus \eta)) \subseteq UNIV - \text{varDiffs}$  using *Eq.premis(2)*  
 by *simp*  
 moreover have  $\llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t a = 0$  using *Eq.premis(1)*  
 by *simp*  
 ultimately have  $(\forall c. (a, c) \in \text{ODEsystem xfList with } G \longrightarrow \llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t c = 0)$   
 using *dInvForTrms-prelim listsHyp* by *blast*  
 hence  $\llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t (F t) = 0$   
 using *tcHyp cHyp* by *simp*  
 from *this* have  $\llbracket \vartheta \rrbracket_t (F t) = \llbracket \eta \rrbracket_t (F t)$   
 by *simp*  
 also have  $(\llbracket \vartheta \doteq \eta \rrbracket_P) c = (\llbracket \vartheta \rrbracket_t (F t) = \llbracket \eta \rrbracket_t (F t))$   
 using *tcHyp* by *simp*  
 ultimately show *?case*  
 by *simp*  
 next  
 case (*Less*  $\vartheta \eta$ )  
 hence  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $0 \leq (\llbracket (\text{map } (vdiff \circ \pi_1) \text{ xfList} \otimes uInput) \upharpoonright \partial_t (\eta \oplus (\ominus \vartheta)) \rrbracket_t st$   
 using *less-pval-to-tval* by *metis*

also from *Less.premis*(2) have  $\text{trmVars } (\eta \oplus (\ominus \vartheta)) \subseteq \text{UNIV} - \text{varDiffs}$   
 by *simp*  
 moreover have  $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t a > 0$   
 using *Less.premis*(1) by *simp*  
 ultimately have  $(\forall c. (a, c) \in \text{ODEsystem } \text{xfList} \text{ with } G \longrightarrow \llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t c > 0)$   
 using *dInvForProps-prelim*(1) *listsHyp* by *blast*  
 hence  $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t (F t) > 0$   
 using *tcHyp cHyp* by *simp*  
 from this have  $\llbracket \eta \rrbracket_t (F t) > \llbracket \vartheta \rrbracket_t (F t)$   
 by *simp*  
 also have  $\llbracket \vartheta \prec \eta \rrbracket_P c = (\llbracket \vartheta \rrbracket_t (F t) < \llbracket \eta \rrbracket_t (F t))$   
 using *tcHyp* by *simp*  
 ultimately show *?case*  
 by *simp*  
 next  
 case (*Leq*  $\vartheta \eta$ )  
 hence  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set } \text{xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $0 \leq (\llbracket (\text{map } (\text{vdiff} \circ \pi_1) \text{xfList} \otimes \text{uInput}) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle \rrbracket_t) st$   
 using *leq-pval-to-tval* by *metis*  
 also from *Leq.premis*(2) have  $\text{trmVars } (\eta \oplus (\ominus \vartheta)) \subseteq \text{UNIV} - \text{varDiffs}$   
 by *simp*  
 moreover have  $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t a \geq 0$   
 using *Leq.premis*(1) by *simp*  
 ultimately have  $(\forall c. (a, c) \in \text{ODEsystem } \text{xfList} \text{ with } G \longrightarrow \llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t c \geq 0)$   
 using *dInvForProps-prelim*(2) *listsHyp* by *blast*  
 hence  $\llbracket \eta \oplus (\ominus \vartheta) \rrbracket_t (F t) \geq 0$   
 using *tcHyp cHyp* by *simp*  
 from this have  $(\llbracket \eta \rrbracket_t (F t) \geq \llbracket \vartheta \rrbracket_t (F t))$   
 by *simp*  
 also have  $\llbracket \vartheta \preceq \eta \rrbracket_P c = (\llbracket \vartheta \rrbracket_t (F t) \leq \llbracket \eta \rrbracket_t (F t))$   
 using *tcHyp* by *simp*  
 ultimately show *?case*  
 by *simp*  
 next  
 case (*And*  $\varphi 1 \varphi 2$ )  
 thus *?case*  
 by *simp*  
 next  
 case (*Or*  $\varphi 1 \varphi 2$ )  
 thus *?case*  
 by *auto*  
 qed  
 qed

theorem *dInv*:

assumes  $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set } \text{xfList} \rrbracket)) \longrightarrow st (\partial str) = 0 \longrightarrow$   
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{xfList}) \otimes \text{uInput}) \upharpoonright \partial_P \varphi \upharpoonright \rrbracket_P st$   
 and  $\text{termVarsHyp}:\text{propVars } \varphi \subseteq (\text{UNIV} - \text{varDiffs})$   
 and  $\text{listsHyp}:\text{map } \pi_2 \text{xfList} = \text{map } \text{tval } \text{uInput}$   
 and  $\text{phi-p}:\text{P} = \llbracket \varphi \rrbracket_P$   
 shows  $\text{PRE } P (\text{ODEsystem } \text{xfList} \text{ with } G) \text{ POST } P$   
 proof(*clarsimp*)  
 fix  $a b$   
 assume  $(a, b) \in \llbracket P \rrbracket$   
 from this have  $a\text{Hyp}:a = b \wedge P a$   
 by (*simp add: p2r-def*)  
 have  $P a \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem } \text{xfList} \text{ with } G) \longrightarrow P c)$   
 using *assms dInv-prelim* by *metis*  
 from this and *aHyp* have  $\forall c. (a, c) \in (\text{ODEsystem } \text{xfList} \text{ with } G) \longrightarrow P c$   
 by *blast*

**thus**  $(a, b) \in wp \text{ (ODEsystem } xfList \text{ with } G) \upharpoonright P$   
**using** *aHyp* **by** (*simp add: boxProgrPred-chrcrtrzn*)  
**qed**

**theorem** *dInvFinal*:

**assumes**  $\forall st. G \ st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket set \ xfList \rrbracket) \longrightarrow st \ (\partial \ str) = 0) \longrightarrow$   
 $\llbracket (map \ (vdiff \circ \pi_1) \ xfList) \otimes uInput \rrbracket \partial_P \varphi \rrbracket_P \ st$   
**and** *termVarsHyp*:  $propVars \ \varphi \subseteq (UNIV - varDiffs)$   
**and** *listsHyp*:  $map \ \pi_2 \ xfList = map \ tval \ uInput$   
**and** *impls*:  $\llbracket P \rrbracket \subseteq \llbracket F \rrbracket \wedge \llbracket F \rrbracket \subseteq \llbracket Q \rrbracket$   
**and** *phi-f*:  $F = \llbracket \varphi \rrbracket_P$   
**shows** *PRE*  $P \text{ (ODEsystem } xfList \text{ with } G) \text{ POST } Q$   
**apply**(*rule-tac*  $C = \llbracket \varphi \rrbracket_P$  **in** *dCut*)  
**apply**(*subgoal-tac*  $\llbracket F \rrbracket \subseteq wp \text{ (ODEsystem } xfList \text{ with } G) \upharpoonright F$ )  
**using** *impls* **and** *phi-f* **apply** *blast*  
**apply**(*subgoal-tac* *PRE*  $F \text{ (ODEsystem } xfList \text{ with } G) \text{ POST } F, \text{ simp}$ )  
**apply**(*rule-tac*  $\varphi = \varphi$  **and**  $uInput = uInput$  **in** *dInv*)  
**prefer** 5 **apply**(*subgoal-tac* *PRE*  $P \text{ (ODEsystem } xfList \text{ with } (\lambda s. G \ s \wedge F \ s)) \text{ POST } Q, \text{ simp add: phi-f}$ )  
**apply**(*rule* *dWeakening*)  
**using** *impls* **apply** *simp*  
**using** *assms* **by** *simp-all*

**end**

**theory** *VC-diffKAD-examples*  
**imports** *VC-diffKAD*

**begin**

### 0.19.5 Rules Testing

In this section we test the recently developed rules with simple dynamical systems.

— Example of hybrid program verified with the rule *dSolve* and a single differential equation:  $x' = v$ .

**lemma** *motion-with-constant-velocity*:

*PRE*  $(\lambda s. s \text{ ''}y'' < s \text{ ''}x'' \wedge s \text{ ''}v'' > 0)$   
 $(ODEsystem \llbracket (x'', (\lambda s. s \text{ ''}v'')) \rrbracket \text{ with } (\lambda s. True))$   
*POST*  $(\lambda s. (s \text{ ''}y'' < s \text{ ''}x''))$   
**apply**(*rule-tac*  $uInput = [\lambda t s. s \text{ ''}v'' \cdot t + s \text{ ''}x'']$  **in** *dSolve-toSolveUBC*)  
**prefer** 9 **subgoal** **by** (*simp add: wp-rel vdiff-def add-strict-increasing2*)  
**apply** (*simp-all add: vdiff-def varDiffs-def*)  
**prefer** 2 **apply** (*simp add: solvesStoreIVP-def vdiff-def varDiffs-def*)  
**apply** (*clarify, rule-tac*  $f'1 = \lambda x. s \text{ ''}v''$  **and**  $g'1 = \lambda x. 0$  **in** *derivative-intros*(189))  
**apply** (*rule-tac*  $f'1 = \lambda x. 0$  **and**  $g'1 = \lambda x. 1$  **in** *derivative-intros*(192))  
**by** (*auto intro: derivative-intros*)

Same hybrid program verified with *dSolve* and the system of ODEs:  $x' = v, v' = a$ . The uniqueness part of the proof requires a preliminary lemma.

**lemma** *flow-vel-is-galilean-vel*:

**assumes** *solHyp*:  $\varphi_s \text{ solvesTheStoreIVP } [(x, \lambda s. s \ v), (v, \lambda s. s \ a)] \text{ withInitState } s$   
**and** *tHyp*:  $r \leq t$  **and** *rHyp*:  $0 \leq r$  **and** *distinct*:  $x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin varDiffs$   
**shows**  $\varphi_s \ r \ v = s \ a \cdot r + s \ v$

**proof**—

**from** *assms* **have** 1:  $(\lambda t. \varphi_s \ t \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a) \ \{0..t\} \ UNIV \wedge \varphi_s \ 0 \ v = s \ v$   
**by** (*simp add: solvesStoreIVP-def*)  
**from** *assms* **have** *obs*:  $\forall r \in \{0..t\}. \varphi_s \ r \ a = s \ a$   
**by** (*auto simp: solvesStoreIVP-def varDiffs-def*)  
**have** 2:  $(\lambda t. s \ a \cdot t + s \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a) \ \{0..t\} \ UNIV$   
**unfolding** *solves-ode-def* **apply**(*subgoal-tac*  $((\lambda x. s \ a \cdot x + s \ v) \text{ has-vderiv-on } (\lambda x. s \ a)) \ \{0..t\}$ )  
**using** *obs* **apply** (*simp add: has-vderiv-on-def*) **by** (*rule galilean-transform*)  
**have** 3: *unique-on-bounded-closed*  $0 \ \{0..t\} \ (s \ v) \ (\lambda t \ r. \varphi_s \ t \ a) \ UNIV$  (*if*  $t = 0$  *then* 1 *else*  $1/(t+1)$ )

```

apply(simp add: ubc-definitions del: comp-apply, rule conjI)
using rHyp tHyp obs apply(simp-all del: comp-apply)
apply(clarify, rule continuous-intros) prefer 3 apply safe
apply(rule continuous-intros)
apply(auto intro: continuous-intros)
by (metis continuous-on-const continuous-on-eq)
thus  $\varphi_s \ r \ v = s \ a \cdot r + s \ v$ 
apply(rule-tac unique-on-bounded-closed.unique-solution[of 0 {0..t} s v
  ( $\lambda t \ r. \varphi_s \ t \ a$ ) UNIV (if  $t = 0$  then 1 else  $1 / (t + 1)$ ) ( $\lambda t. \varphi_s \ t \ v$ )])
using rHyp tHyp 1 2 and 3 by auto
qed

```

**lemma** *motion-with-constant-acceleration*:

```

PRE ( $\lambda s. s \ "y" < s \ "x" \wedge s \ "v" \geq 0 \wedge s \ "a" > 0$ )
  (ODEsystem [( $"x", (\lambda s. s \ "v")$ ), ( $"v", (\lambda s. s \ "a")$ )] with ( $\lambda s. \text{True}$ ))
POST ( $\lambda s. (s \ "y" < s \ "x")$ )
apply(rule-tac uInput=[ $\lambda t \ s. s \ "a" \cdot t^2/2 + s \ "v" \cdot t + s \ "x",$ 
 $\lambda t \ s. s \ "a" \cdot t + s \ "v"$ ]) in dSolve-toSolveUBC)
prefer 9 subgoal by(simp add: wp-rel vdiff-def add-strict-increasing2)
prefer 6 subgoal
  apply(simp add: vdiff-def, clarify, rule conjI)
  by(rule galilean-transform)+
prefer 6 subgoal
  apply(simp add: vdiff-def, safe)
  by(rule continuous-intros)+
prefer 6 subgoal
  apply(simp add: vdiff-def, safe)
  subgoal for  $s \ \varphi_s \ t \ r$  apply(rule flow-vel-is-galilean-vel[of  $\varphi_s \ "x" \dots t$ ])
    by(simp-all add: varDiffs-def vdiff-def)
  apply(simp add: solvesStoreIVP-def vdiff-def varDiffs-def) done
by (auto simp: varDiffs-def vdiff-def)

```

Example of a hybrid system with two modes verified with the equality dS. We also need to provide a previous (similar) lemma.

**lemma** *flow-vel-is-galilean-vel2*:

```

assumes solHyp: $\varphi_s$  solvesTheStoreIVP [( $x, \lambda s. s \ v$ ), ( $v, \lambda s. -s \ a$ )] withInitState  $s$ 
and tHyp: $r \leq t$  and rHyp: $0 \leq r$  and distinct: $x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin \text{varDiffs}$ 
shows  $\varphi_s \ r \ v = s \ v - s \ a \cdot r$ 
proof—
from assms have 1:(( $\lambda t. \varphi_s \ t \ v$ ) solves-ode ( $\lambda t \ r. -\varphi_s \ t \ a$ )) {0..t} UNIV  $\wedge \varphi_s \ 0 \ v = s \ v$ 
  by (simp add: solvesStoreIVP-def)
from assms have obs: $\forall r \in \{0..t\}. \varphi_s \ r \ a = s \ a$ 
  by(auto simp: solvesStoreIVP-def varDiffs-def)
have 2:(( $\lambda t. -s \ a \cdot t + s \ v$ ) solves-ode ( $\lambda t \ r. -\varphi_s \ t \ a$ )) {0..t} UNIV
  unfolding solves-ode-def apply(subgoal-tac (( $\lambda x. -s \ a \cdot x + s \ v$ ) has-vderiv-on ( $\lambda x. -s \ a$ )) {0..t})
  using obs apply (simp add: has-vderiv-on-def) by(rule galilean-transform)
have 3:unique-on-bounded-closed 0 {0..t} ( $s \ v$ ) ( $\lambda t \ r. -\varphi_s \ t \ a$ ) UNIV (if  $t = 0$  then 1 else  $1/(t+1)$ )
  apply(simp add: ubc-definitions del: comp-apply, rule conjI)
  using rHyp tHyp obs apply(simp-all del: comp-apply)
  apply(clarify, rule continuous-intros) prefer 3 apply safe
  apply(rule continuous-intros)+
  apply(auto intro: continuous-intros)
  by (metis continuous-on-const continuous-on-eq)
thus  $\varphi_s \ r \ v = s \ v - s \ a \cdot r$ 
  apply(rule-tac unique-on-bounded-closed.unique-solution[of 0 {0..t} s v
    ( $\lambda t \ r. -\varphi_s \ t \ a$ ) UNIV (if  $t = 0$  then 1 else  $1 / (t + 1)$ ) ( $\lambda t. \varphi_s \ t \ v$ )])
  using rHyp tHyp 1 2 and 3 by auto
qed

```



**lemma** *single-hop-ball*:

```

PRE (λ s. 0 ≤ s "x" ∧ s "x" = H ∧ s "v" = 0 ∧ s "g" > 0 ∧ 1 ≥ c ∧ c ≥ 0)
  (((ODEsystem [(("x", λ s. s "v"), ("v", λ s. - s "g")) with (λ s. 0 ≤ s "x")));
   (IF (λ s. s "x" = 0) THEN ("v" ::= (λ s. - c · s "v")) ELSE ("v" ::= (λ s. s "v"))))
  POST (λ s. 0 ≤ s "x" ∧ s "x" ≤ H)
apply(simp, subst dS[of [λ t s. - s "g" · t ^ 2/2 + s "v" · t + s "x", λ t s. - s "g" · t + s "v"]])
  — Given solution is actually a solution.
apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton, safe)
apply(rule galilean-transform-eq, simp)+
apply(rule galilean-transform)+
  — Uniqueness of the flow.
apply(rule ubcStoreUniqueSol, simp)
apply(simp add: vdiff-def del: comp-apply)
apply(auto intro: continuous-intros del: comp-apply)[1]
apply(rule continuous-intros)+
apply(simp add: vdiff-def, safe)
apply(clarsimp) subgoal for s X t τ
  apply(rule flow-vel-is-galilean-vel2[of X "x"])
  by(simp-all add: varDiffs-def vdiff-def)
apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def)
apply(simp add: vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def
  has-vderiv-on-singleton galilean-transform-eq galilean-transform)
  — Relation Between the guard and the postcondition.
by(auto simp: vdiff-def p2r-def)

```

— Example of hybrid program verified with differential weakening.

**lemma** *system-where-the-guard-implies-the-postcondition*:

```

PRE (λ s. s "x" = 0)
  (ODEsystem [(("x", λ s. s "x" + 1))] with (λ s. s "x" ≥ 0))
  POST (λ s. s "x" ≥ 0)
using dWeakening by blast

```

**lemma** *system-where-the-guard-implies-the-postcondition2*:

```

PRE (λ s. s "x" = 0)
  (ODEsystem [(("x", λ s. s "x" + 1))] with (λ s. s "x" ≥ 0))
  POST (λ s. s "x" ≥ 0)
apply(simp add: wp-rel)
by (auto simp: relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def)

```

— Example of system proved with a differential invariant.

**lemma** *circular-motion*:

```

PRE (λ s. (s "x") · (s "x") + (s "y") · (s "y") - (s "r") · (s "r") = 0)
  (ODEsystem [(("x", λ s. s "y"), ("y", λ s. - s "x")) with G)
  POST (λ s. (s "x") · (s "x") + (s "y") · (s "y") - (s "r") · (s "r") = 0)
apply(rule-tac η=(t_V "x")⊙(t_V "x") ⊕ (t_V "y")⊙(t_V "y") ⊕ (⊖(t_V "r")⊙(t_V "r")))
  and uInput=[t_V "y", ⊖ (t_V "x")] in dInvForTrms)
apply(simp-all add: vdiff-def varDiffs-def)
apply(clarsimp, erule-tac x="r" in allE)
by simp

```

— Example of systems proved with differential invariants, cuts and weakenings.

**lemma** *motion-with-constant-velocity-and-invariants*:

```

PRE (λ s. s "x" > s "y" ∧ s "v" > 0)
  (ODEsystem [(("x", λ s. s "v")) with (λ s. True))
  POST (λ s. s "x" > s "y")
apply(rule-tac C = λ s. s "v" > 0 in dCut)
apply(rule-tac φ = (t_C 0) < (t_V "v") and uInput=[t_V "v"] in dInvFinal)
apply(simp-all add: vdiff-def varDiffs-def, clarify, erule-tac x="v" in allE, simp)
apply(rule-tac C = λ s. s "x" > s "y" in dCut)

```

**apply**(rule-tac  $\varphi = (t_V \text{ ''y''}) \prec (t_V \text{ ''x''})$  **and**  $uInput = [t_V \text{ ''v''}]$  **and**  
 $F = \lambda s. s \text{ ''x''} > s \text{ ''y''}$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $vdiff\text{-}def \text{ varDiffs-}def$ , clarify, erule-tac  $x = \text{''y''}$  **in**  $allE$ , simp)  
**using**  $dWeakening$  **by** simp

**lemma** *motion-with-constant-acceleration-and-invariants*:

$PRE (\lambda s. s \text{ ''y''} < s \text{ ''x''} \wedge s \text{ ''v''} \geq 0 \wedge s \text{ ''a''} > 0)$   
 $((ODEsystem [( \text{''x''}, (\lambda s. s \text{ ''v''}), (\text{''v''}, (\lambda s. s \text{ ''a''})) ] \text{ with } (\lambda s. True)))$   
 $POST (\lambda s. (s \text{ ''y''} < s \text{ ''x''}))$   
**apply**(rule-tac  $C = \lambda s. s \text{ ''a''} > 0$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_C 0) \prec (t_V \text{ ''a''})$  **and**  $uInput = [t_V \text{ ''v''}, t_V \text{ ''a''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $vdiff\text{-}def \text{ varDiffs-}def$ , clarify, erule-tac  $x = \text{''a''}$  **in**  $allE$ , simp)  
**apply**(rule-tac  $C = \lambda s. s \text{ ''v''} \geq 0$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_C 0) \preceq (t_V \text{ ''v''})$  **and**  $uInput = [t_V \text{ ''v''}, t_V \text{ ''a''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $vdiff\text{-}def \text{ varDiffs-}def$ )  
**apply**(rule-tac  $C = \lambda s. s \text{ ''x''} > s \text{ ''y''}$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_V \text{ ''y''}) \prec (t_V \text{ ''x''})$  **and**  $uInput = [t_V \text{ ''v''}, t_V \text{ ''a''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $varDiffs\text{-}def \text{ vdiff-}def$ , clarify, erule-tac  $x = \text{''y''}$  **in**  $allE$ , simp)  
**using**  $dWeakening$  **by** simp

— We revisit the two modes example from before, and prove it with invariants.

**lemma** *single-hop-ball-and-invariants*:

$PRE (\lambda s. 0 \leq s \text{ ''x''} \wedge s \text{ ''x''} = H \wedge s \text{ ''v''} = 0 \wedge s \text{ ''g''} > 0 \wedge 1 \geq c \wedge c \geq 0)$   
 $((ODEsystem [( \text{''x''}, (\lambda s. s \text{ ''v''}), (\text{''v''}, (\lambda s. - s \text{ ''g''})) ] \text{ with } (\lambda s. 0 \leq s \text{ ''x''})));$   
 $(IF (\lambda s. s \text{ ''x''} = 0) THEN (\text{''v''} ::= (\lambda s. - c \cdot s \text{ ''v''})) ELSE (\text{''v''} ::= (\lambda s. s \text{ ''v''}))))$   
 $POST (\lambda s. 0 \leq s \text{ ''x''} \wedge s \text{ ''x''} \leq H)$   
**apply**(simp, rule-tac  $C = \lambda s. s \text{ ''g''} > 0$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_C 0) \prec (t_V \text{ ''g''})$  **and**  $uInput = [t_V \text{ ''v''}, \ominus t_V \text{ ''g''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $vdiff\text{-}def \text{ varDiffs-}def$ , clarify, erule-tac  $x = \text{''g''}$  **in**  $allE$ , simp)  
**apply**(rule-tac  $C = \lambda s. s \text{ ''v''} \leq 0$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_V \text{ ''v''}) \preceq (t_C 0)$  **and**  $uInput = [t_V \text{ ''v''}, \ominus t_V \text{ ''g''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $vdiff\text{-}def \text{ varDiffs-}def$ )  
**apply**(rule-tac  $C = \lambda s. s \text{ ''x''} \leq H$  **in**  $dCut$ )  
**apply**(rule-tac  $\varphi = (t_V \text{ ''x''}) \preceq (t_C H)$  **and**  $uInput = [t_V \text{ ''v''}, \ominus t_V \text{ ''g''}]$  **in**  $dInvFinal$ )  
**apply**(simp-all add:  $varDiffs\text{-}def \text{ vdiff-}def$ )  
**using**  $dWeakening$  **by** simp

— Finally, we add a well known example in the hybrid systems community, the bouncing ball.

**lemma** *bouncing-ball-invariant*:  $0 \leq x \implies 0 < g \implies 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \implies (x :: real) \leq H$   
**proof**—

**assume**  $0 \leq x$  **and**  $0 < g$  **and**  $2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v$   
**then have**  $v \cdot v = 2 \cdot g \cdot H - 2 \cdot g \cdot x \wedge 0 < g$  **by** *auto*  
**hence**  $*: v \cdot v = 2 \cdot g \cdot (H - x) \wedge 0 < g \wedge v \cdot v \geq 0$   
**using** *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)  
**from this have**  $(v \cdot v) / (2 \cdot g) = (H - x)$  **by** *auto*  
**also from**  $*$  **have**  $(v \cdot v) / (2 \cdot g) \geq 0$   
**using** *divide-nonneg-pos* **by** *fastforce*  
**ultimately have**  $H - x \geq 0$  **by** *linarith*  
**thus** *?thesis* **by** *auto*

**qed**

**lemma** *bouncing-ball*:

$PRE (\lambda s. 0 \leq s \text{ ''x''} \wedge s \text{ ''x''} = H \wedge s \text{ ''v''} = 0 \wedge s \text{ ''g''} > 0)$   
 $((ODEsystem [( \text{''x''}, (\lambda s. s \text{ ''v''}), (\text{''v''}, (\lambda s. - s \text{ ''g''})) ] \text{ with } (\lambda s. 0 \leq s \text{ ''x''})));$   
 $(IF (\lambda s. s \text{ ''x''} = 0) THEN (\text{''v''} ::= (\lambda s. - s \text{ ''v''})) ELSE Id))^*$   
 $POST (\lambda s. 0 \leq s \text{ ''x''} \wedge s \text{ ''x''} \leq H)$   
**apply**(rule *wp-loopI* [*of* -  $\lambda s. 0 \leq s \text{ ''x''} \wedge 0 < s \text{ ''g''} \wedge$   
 $2 \cdot s \text{ ''g''} \cdot s \text{ ''x''} = 2 \cdot s \text{ ''g''} \cdot H - (s \text{ ''v''} \cdot s \text{ ''v''})$ ])  
**apply**(simp, *clarsimp simp: bouncing-ball-invariant*, simp)  
**apply**(rule-tac  $C = \lambda s. s \text{ ''g''} > 0$  **in**  $dCut$ )

```

apply(rule-tac  $\varphi = ((t_C \ 0) \prec (t_V \ "g"))$ ) and  $uInput=[t_V \ "v", \ominus t_V \ "g"]$  in  $dInvFinal$ )
  apply(simp-all add:  $vdiff$ -def  $varDiffs$ -def, clarify, erule-tac  $x="g"$  in  $allE$ , simp)
apply(rule-tac  $C = \lambda s. 2 \cdot s \ "g" \cdot s \ "x" = 2 \cdot s \ "g" \cdot H - s \ "v" \cdot s \ "v"$  in  $dCut$ )
  apply(rule-tac  $\varphi = (t_C \ 2) \odot (t_V \ "g") \odot (t_C \ H) \oplus (\ominus ((t_V \ "v") \odot (t_V \ "v")))$ )
 $\doteq (t_C \ 2) \odot (t_V \ "g") \odot (t_V \ "x")$  and  $uInput=[t_V \ "v", \ominus t_V \ "g"]$  in  $dInvFinal$ )
  apply(simp-all add:  $vdiff$ -def  $varDiffs$ -def, clarify, erule-tac  $x="g"$  in  $allE$ , simp)
by (rule  $dWeakening$ ,  $clarsimp$ )

end

```