

CPSVerification

CPSVerification

November 21, 2019

Contents

0.1	Hybrid Systems Preliminaries	6
0.1.1	Functions	6
0.1.2	Orders	6
0.1.3	Real numbers	8
0.1.4	Single variable derivatives	9
0.1.5	Filters	12
0.1.6	Multivariable derivatives	13
0.2	Ordinary Differential Equations	15
0.2.1	Initial value problems and orbits	15
0.2.2	Differential Invariants	17
0.2.3	Picard-Lindelof	20
0.2.4	Flows for ODEs	22
0.3	Linear algebra for hybrid systems	28
0.3.1	Properties of some vector operations	28
0.3.2	Matrix norms	30
0.3.3	Picard Lindelof for linear systems	34
0.3.4	Diagonalization	35
0.3.5	Squared matrices	40
0.3.6	Flow for squared matrix systems	51
0.4	Verification components for hybrid systems	52
0.4.1	Verification of regular programs	52
0.4.2	Verification of hybrid programs	55
0.4.3	Derivation of the rules of dL	57
0.4.4	Examples	60
0.5	Verification components with predicate transformers	70
0.5.1	Verification of regular programs	71
0.5.2	Verification of hybrid programs	73
0.5.3	Derivation of the rules of dL	75
0.5.4	Examples	78
0.6	Verification components with Kleene Algebras	86
0.6.1	Hoare logic and refinement in KAT	86
0.6.2	refinement KAT	89
0.6.3	Verification in AKA (KAD)	91
0.6.4	Relational model	92

0.6.5	State transformer model	93
0.7	Verification components with relational MKA	96
0.7.1	Store and weakest preconditions	96
0.7.2	Verification of hybrid programs	97
0.7.3	Derivation of the rules of dL	99
0.8	Verification components with MKA and non-deterministic func- tions	102
0.8.1	Store and weakest preconditions	102
0.8.2	Verification of hybrid programs	104
0.8.3	Derivation of the rules of dL	107
0.8.4	Examples	110
0.9	Verification and refinement of HS in the relational KAT . . .	117
0.9.1	Store and Hoare triples	117
0.9.2	Verification of hybrid programs	120
0.9.3	Refinement Components	123
0.9.4	Derivation of the rules of dL	126
0.9.5	Examples	127
0.10	Verification and refinement of HS in the relational KAT . . .	141
0.10.1	Store and Hoare triples	141
0.10.2	Verification of hybrid programs	143
0.10.3	Refinement Components	146
0.10.4	Derivation of the rules of dL	150
0.10.5	Examples	151
0.11	Kleene Algebras	164
0.11.1	Left Near Kleene Algebras	164
0.11.2	Left Pre-Kleene Algebras	169
0.11.3	Left Kleene Algebras	175
0.11.4	Left Kleene Algebras with Zero	178
0.11.5	Pre-Kleene Algebras	178
0.11.6	Kleene Algebras	179
0.12	Models of Dioids	183
0.12.1	The Powerset Dioid over a Monoid	183
0.12.2	Language Dioids	183
0.12.3	Relation Dioids	184
0.12.4	Trace Dioids	184
0.12.5	Sets of Traces	185
0.12.6	The Path Dioid	186
0.12.7	Path Models with the Empty Path	187
0.12.8	Path Models without the Empty Path	190
0.12.9	The Distributive Lattice Dioid	192
0.12.10	The Boolean Dioid	193
0.12.11	The Max-Plus Dioid	193
0.12.12	The Min-Plus Dioid	195
0.13	Models of Kleene Algebras	198

0.13.1	Preliminary Lemmas	198
0.13.2	The Powerset Kleene Algebra over a Monoid	199
0.13.3	Relation Kleene Algebras	199
0.13.4	Trace Kleene Algebras	200
0.13.5	Path Kleene Algebras	201
0.13.6	The Distributive Lattice Kleene Algebra	203
0.13.7	The Min-Plus Kleene Algebra	203
0.14	Domain Semirings	204
0.14.1	Domain Semigroups and Domain Monoids	204
0.14.2	Domain Near-Semirings	208
0.14.3	Domain Pre-Dioids	213
0.14.4	Domain Semirings	215
0.14.5	The Algebra of Domain Elements	216
0.14.6	Domain Semirings with a Greatest Element	217
0.14.7	Forward Diamond Operators	217
0.14.8	Domain Kleene Algebras	219
0.15	Antidomain Semirings	221
0.15.1	Antidomain Monoids	221
0.15.2	Antidomain Near-Semirings	226
0.15.3	Antidomain Pre-Dioids	231
0.15.4	Antidomain Semirings	236
0.15.5	The Boolean Algebra of Domain Elements	238
0.15.6	Further Properties	239
0.15.7	Forward Box and Diamond Operators	242
0.15.8	Antidomain Kleene Algebras	245
0.16	Range and Antirange Semirings	247
0.16.1	Range Semirings	247
0.16.2	Antirange Semirings	248
0.16.3	Antirange Kleene Algebras	249
0.17	Modal Kleene Algebras	250
0.18	Models of Modal Kleene Algebras	252
0.19	Components Based on Kleene Algebra with Domain	253
0.19.1	Verification Component for Backward Reasoning	254
0.20	VC_diffKAD	259
0.20.1	Stack Theories Preliminaries: VC_KAD and ODEs	259
0.20.2	VC_diffKAD Preliminaries	261
0.20.3	Phase Space Relational Semantics	272
0.20.4	Derivation of Differential Dynamic Logic Rules	274
0.20.5	Rules Testing	291

0.1 Hybrid Systems Preliminaries

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

theory *hs-prelims*

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

notation *has-derivative* $((1(D \mapsto (-)) / -) [65,65] 61)$

notation *has-vderiv-on* $((1 D - = (-) / \text{on } -) [65,65] 61)$

notation *norm* $((1 \|\cdot\|) [65] 61)$

0.1.1 Functions

lemma *case-of-fst[simp]*: $(\lambda x. \text{case } x \text{ of } (t, x) \Rightarrow f t) = (\lambda x. (f \circ \text{fst}) x)$
by *auto*

lemma *case-of-snd[simp]*: $(\lambda x. \text{case } x \text{ of } (t, x) \Rightarrow f x) = (\lambda x. (f \circ \text{snd}) x)$
by *auto*

0.1.2 Orders

lemma *finite-image-of-finite[simp]*:
fixes $f :: 'a :: \text{finite} \Rightarrow 'b$
shows *finite* $\{x. \exists i. x = f i\}$
using *finite-Atleast-Atmost-nat* **by** *force*

lemma *le-max-image-of-finite[simp]*:
fixes $f :: 'a :: \text{finite} \Rightarrow 'b :: \text{linorder}$
shows $(f i) \leq \text{Max } \{x. \exists i. x = f i\}$
by $(\text{rule } \text{Max.coboundedI}, \text{simp-all}) (\text{rule-tac } x=i \text{ in } \text{exI}, \text{simp})$

lemma *cSup-eq-linorder*:
fixes $c :: 'a :: \text{conditionally-complete-linorder}$
assumes $X \neq \{\}$ **and** $\forall x \in X. x \leq c$
and *bdd-above* X **and** $\forall y < c. \exists x \in X. y < x$
shows $\text{Sup } X = c$
by $(\text{meson } \text{assms } \text{cSup-least } \text{less-cSup-iff } \text{less-le})$

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 $(\text{metis } \text{assms } \text{cSup-eq-maximum } \text{order-class.order.antisym})$

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 $(\text{rule } \text{cSup-eq}, \text{auto})$

lemma *cSup-finite-ex*:

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*:

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 *bdd-above-ltimes*:

fixes $c :: 'a :: \text{linordered-ring-strict}$
assumes $c \geq 0$ **and** *bdd-above* X
shows *bdd-above* $\{c * x \mid x. x \in X\}$
using *assms unfolding bdd-above-def apply clarsimp*
apply (*rule-tac x=c * M in exI, clarsimp*)
using *mult-left-mono* **by** *blast*

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

lemma *suminfI*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{t2-space, comm-monoid-add}\}$
shows $f \text{ sums } k \implies \text{suminf } f = k$
unfolding *sums-iff* **by** *simp*

lemma *suminf-eq-sum*:

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

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]*)

0.1.3 Real numbers

lemma *ge-one-sqrt-le*: $1 \leq x \implies \text{sqrt } x \leq x$

by (*metis basic-trans-rules(23) monoid-mult-class.power2-eq-square more-arith-simps(6)*)

mult-left-mono real-sqrt-le-iff' zero-le-one)

lemma *sqrt-real-nat-le*: $\text{sqrt } (\text{real } n) \leq \text{real } n$

by (*metis (full-types) abs-of-nat le-square of-nat-mono of-nat-mult real-sqrt-abs2 real-sqrt-le-iff*)

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 *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 linarith*

lemma *real-ivl-egs*:

assumes $0 < r$

shows $\text{ball } x\ r = \{x - r < \dots < x + r\}$ **and** $\{x - r < \dots < x + r\} = \{x - r < \dots < x + r\}$

and $\text{ball } (r / 2)\ (r / 2) = \{0 < \dots < r\}$ **and** $\{0 < \dots < r\} = \{0 < \dots < r\}$

and $\text{ball } 0\ r = \{-r < \dots < r\}$ **and** $\{-r < \dots < r\} = \{-r < \dots < r\}$

and $\text{cball } x\ r = \{x - r \dots x + r\}$ **and** $\{x - r \dots x + r\} = \{x - r \dots x + r\}$

and $\text{cball } (r / 2)\ (r / 2) = \{0 \dots r\}$ **and** $\{0 \dots r\} = \{0 \dots r\}$

and $\text{cball } 0\ r = \{-r \dots r\}$ **and** $\{-r \dots r\} = \{-r \dots r\}$

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

using *assms* **apply** (*auto simp: cball-def ball-def dist-norm*)

by (*simp-all add: field-simps*)

lemma *norm-rotate-simps*[*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(*simp add: power2-diff power-mult-distrib*)
also have $(x * \sin t + y * \cos t)^2 = y^2 * (\cos t)^2 + x^2 * (\sin t)^2 + 2 * (x * \cos t) * (y * \sin t)$
by(*simp add: power2-sum power-mult-distrib*)
ultimately show $(x * \cos t - y * \sin t)^2 + (x * \sin t + y * \cos t)^2 = x^2 + y^2$

by (*simp add: Groups.mult-ac(2) Groups.mult-ac(3) right-diff-distrib sin-squared-eq*)

thus $(x * \cos t + y * \sin t)^2 + (y * \cos t - x * \sin t)^2 = x^2 + y^2$
by (*simp add: add.commute add.left-commute power2-diff power2-sum*)
qed

0.1.4 Single variable derivatives

— Theorems in the list below are shaped like those on “*derivative_eq_intros*”.

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 *derivative-intros(191)* [*poly-derivatives*]
and *derivative-intros(192)* [*poly-derivatives*]
and *derivative-intros(194)* [*poly-derivatives*]

Below, we consistently name lemmas showing that f' is the derivative of f by starting with “*has...*”. Moreover, if they use the predicate “*has_derivative_at*”, we add them to the list “*derivative_intros*”. Otherwise, if lemmas have an implicit g where $g = f'$, we start their names with “*vderiv*” and end them with “*intro*”.

lemma *has-derivative-exp-scaleRl*[*derivative-intros*]:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $D f \mapsto f'$ at t within T
shows $D (\lambda t. \exp (f t *_{\mathbb{R}} A)) \mapsto (\lambda h. f' h *_{\mathbb{R}} (\exp (f t *_{\mathbb{R}} A) * A))$ at t within T
proof –
from *assms* **have** *bounded-linear f'* **by** *auto*
with *real-bounded-linear* **obtain** m **where** $f': f' = (\lambda h. h * m)$ **by** *blast*
show *?thesis*
using *vector-diff-chain-within[OF - exp-scaleR-has-vector-derivative-right, of f m t T A]*
assms f' **by** (*auto simp: has-vector-derivative-def o-def*)
qed

lemma *has-vector-derivative-mult-const*[*derivative-intros*]:

$((*) a \text{ has-vector-derivative } a) F$
by (*auto intro: derivative-eq-intros*)

lemma *has-derivative-mult-const*[*derivative-intros*]: $D (*) a \mapsto (\lambda t. t *_R a) F$
using *has-vector-derivative-mult-const* **unfolding** *has-vector-derivative-def* **by** *simp*

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

lemma *has-vderiv-on-mult-const*: $D (*) a = (\lambda t. a) \text{ on } T$
using *has-vector-derivative-mult-const* **unfolding** *has-vderiv-on-def* **by** *auto*

lemma *has-vderiv-on-divide-cnst*: $a \neq 0 \implies D (\lambda t. t/a) = (\lambda t. 1/a) \text{ on } T$
unfolding *has-vderiv-on-def* *has-vector-derivative-def* **apply** *clarify*
apply(*rule-tac f'1= $\lambda t. t$ and g'1= $\lambda x. 0$ in derivative-eq-intros(18)*)
by(*auto intro: derivative-eq-intros*)

lemma *has-vderiv-on-power*: $n \geq 1 \implies D (\lambda t. t ^ n) = (\lambda t. n * (t ^ (n - 1)))$
on T
unfolding *has-vderiv-on-def* *has-vector-derivative-def* **apply** *clarify*
by(*rule-tac f'1= $\lambda t. t$ in derivative-eq-intros(15)*) *auto*

lemma *has-vderiv-on-exp*: $D (\lambda t. \exp t) = (\lambda t. \exp t) \text{ on } T$
unfolding *has-vderiv-on-def* *has-vector-derivative-def* **by** (*auto intro: derivative-intros*)

lemma *has-vderiv-on-cos-comp*:
 $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T \implies D (\lambda t. \cos (f t)) = (\lambda t. - (f' t) * \sin (f t))$
on T
apply(*rule vderiv-on-compose-intro[of $\lambda t. \cos t$]*)
unfolding *has-vderiv-on-def* *has-vector-derivative-def* **apply** *clarify*
by(*auto intro!: derivative-eq-intros simp: fun-eq-iff*)

lemma *has-vderiv-on-sin-comp*:
 $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T \implies D (\lambda t. \sin (f t)) = (\lambda t. (f' t) * \cos (f t)) \text{ on } T$
apply(*rule vderiv-on-compose-intro[of $\lambda t. \sin t$]*)
unfolding *has-vderiv-on-def* *has-vector-derivative-def* **apply** *clarify*
by(*auto intro!: derivative-eq-intros simp: fun-eq-iff*)

lemma *has-vderiv-on-exp-comp*:
 $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T \implies D (\lambda t. \exp (f t)) = (\lambda t. (f' t) * \exp (f t)) \text{ on } T$
apply(*rule vderiv-on-compose-intro[of $\lambda t. \exp t$]*)
by (*rule has-vderiv-on-exp, simp-all add: mult.commute*)

lemma *has-vderiv-on-exp-scaleRl*:

assumes $D f = f' \text{ on } T$
shows $D (\lambda x. \exp (f x *_R A)) = (\lambda x. f' x *_R \exp (f x *_R A) * A) \text{ on } T$
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 *vderiv-uminus-intro*[*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-div-cnst-intro*[*poly-derivatives*]:

assumes $(a::\text{real}) \neq 0$ **and** $D f = f' \text{ on } T$ **and** $g = (\lambda t. (f' t)/a)$
shows $D (\lambda t. (f t)/a) = g \text{ on } T$
apply(*rule vderiv-on-compose-intro*[*of* $\lambda t. t/a \lambda t. 1/a$])
using *assms* **by**(*auto intro: has-vderiv-on-divide-cnst*)

lemma *vderiv-npow-intro*[*poly-derivatives*]:

fixes $f::\text{real} \Rightarrow \text{real}$
assumes $n \geq 1$ **and** $D f = f' \text{ on } T$ **and** $g = (\lambda t. n * (f' t) * (f t)^{(n-1)})$
shows $D (\lambda t. (f t)^n) = g \text{ on } T$
apply(*rule vderiv-on-compose-intro*[*of* $\lambda t. t^n$])
using *assms*(1) **apply**(*rule has-vderiv-on-power*)
using *assms* **by** *auto*

lemma *vderiv-cos-intro*[*poly-derivatives*]:

assumes $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T$ **and** $g = (\lambda t. - (f' t) * \sin (f t))$
shows $D (\lambda t. \cos (f t)) = g \text{ on } T$
using *assms* **and** *has-vderiv-on-cos-comp* **by** *auto*

lemma *vderiv-sin-intro*[*poly-derivatives*]:

assumes $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T$ **and** $g = (\lambda t. (f' t) * \cos (f t))$
shows $D (\lambda t. \sin (f t)) = g \text{ on } T$
using *assms* **and** *has-vderiv-on-sin-comp* **by** *auto*

lemma *vderiv-exp-intro*[*poly-derivatives*]:

assumes $D (f::\text{real} \Rightarrow \text{real}) = f' \text{ on } T$ **and** $g = (\lambda t. (f' t) * \exp (f t))$
shows $D (\lambda t. \exp (f t)) = g \text{ on } T$
using *assms* **and** *has-vderiv-on-exp-comp* **by** *auto*

lemma *vderiv-on-exp-scaleRl-intro*[*poly-derivatives*]:

assumes $D f = f' \text{ on } T$ **and** $g' = (\lambda x. f' x *_R \exp (f x *_R A) * A)$
shows $D (\lambda x. \exp (f x *_R A)) = g' \text{ on } T$
using *has-vderiv-on-exp-scaleRl* *assms* **by** *simp*

— Automatically generated derivative rules from this subsection:

thm *derivative-eq-intros*(142,143,144)

— Examples for checking derivatives

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

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

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

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
apply (*intro poly-derivatives*)
using *poly-derivatives(1,2)* **by** *force+*

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
apply (*intro poly-derivatives*)
using *poly-derivatives(1,2)* **by** *force+*

0.1.5 Filters

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

lemma *netlimit-at-within-mono*:
fixes $t::'a::\{\text{perfect-space}, \text{t2-space}\}$
assumes $t \in \text{interior } T$ **and** $T \subseteq S$
shows *netlimit* (at t within S) = t
using *assms(1) interior-mono[OF $T \subseteq S$] 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  $P :: ('a::finite) \Rightarrow 'b \Rightarrow bool$ 
assumes  $h:\forall i. eventually (P\ i)\ F$ 
shows  $eventually (\lambda t. \forall i. P\ i\ t)\ F$ 
proof(unfold eventually-def)
  let  $?F = Rep-filter\ F$ 
  have  $obs: \forall i. ?F\ (P\ i)$ 
    using  $h$  by auto
  have  $?F\ (\lambda t. \forall i \in UNIV. P\ i\ t)$ 
    apply(rule finite-induct)
    by(auto intro: eventually-conj simp: obs h)
  thus  $?F\ (\lambda t. \forall i. P\ i\ t)$ 
    by simp
qed

```

```

lemma eventually-all-finite-mono:
  fixes  $P :: ('a::finite) \Rightarrow 'b \Rightarrow bool$ 
  assumes  $h1: \forall i. eventually (P\ i)\ F$ 
    and  $h2: \forall t. (\forall i. (P\ i\ t)) \longrightarrow Q\ t$ 
  shows  $eventually\ Q\ F$ 
proof–
  have  $eventually (\lambda t. \forall i. P\ i\ t)\ F$ 
    using  $h1$  eventually-all-finite2 by blast
  thus  $eventually\ Q\ F$ 
    unfolding eventually-def
    using  $h2$  eventually-mono by auto
qed

```

0.1.6 Multivariable derivatives

```

lemma frechet-vec-lambda:
  fixes  $f::real \Rightarrow ('a::banach) ^ ('m::finite)$ 
  defines  $m \equiv real\ CARD('m)$ 
  assumes  $\forall i. ((\lambda x. (f\ x\ \$\ i - f\ x_0\ \$\ i - (x - x_0) *_R f'\ t\ \$\ i) /_R (\|x - x_0\|))$ 
     $\longrightarrow 0) (at\ t\ within\ T)$ 
  shows  $((\lambda x. (f\ x - f\ x_0 - (x - x_0) *_R f'\ t) /_R (\|x - x_0\|)) \longrightarrow 0) (at\ t$ 
     $within\ T)$ 
proof(simp add: tendsto-iff, clarify)
  fix  $\varepsilon::real$  assume  $0 < \varepsilon$ 
  let  $? \Delta = \lambda x. x - x_0$  and  $? \Delta f = \lambda x. f\ x - f\ x_0$ 
  let  $?P = \lambda i\ e\ x. inverse\ |\Delta\ x| * (\|f\ x\ \$\ i - f\ x_0\ \$\ i - \Delta\ x *_R f'\ t\ \$\ i\|) < e$ 
    and  $?Q = \lambda x. inverse\ |\Delta\ x| * (\|\Delta f\ x - \Delta\ x *_R f'\ t\|) < \varepsilon$ 
  have  $0 < \varepsilon / sqrt\ m$ 
    using  $\langle 0 < \varepsilon \rangle$  by (auto simp: assms)
  hence  $\forall i. eventually (\lambda x. ?P\ i\ (\varepsilon / sqrt\ m)\ x) (at\ t\ within\ T)$ 
    using assms unfolding tendsto-iff by simp
  thus  $eventually\ ?Q (at\ t\ within\ T)$ 
proof(rule eventually-all-finite-mono, simp add: norm-vec-def L2-set-def, clarify)
  fix  $x::real$ 
  let  $?c = inverse\ |x - x_0|$  and  $?u\ x = \lambda i. f\ x\ \$\ i - f\ x_0\ \$\ i - \Delta\ x *_R f'\ t\ \$\ i$ 

```

assume $hyp: \forall i. ?c * (\| ?u \ x \ i \|) < \varepsilon / \text{sqrt } m$
hence $\forall i. (?c *_R (\| ?u \ x \ i \|^2) < (\varepsilon / \text{sqrt } m)^2$
by (*simp add: power-strict-mono*)
hence $\forall i. ?c^2 * ((\| ?u \ x \ i \|^2) < \varepsilon^2 / m$
by (*simp add: power-mult-distrib power-divide assms*)
hence $\forall i. ?c^2 * ((\| ?u \ x \ i \|^2) < \varepsilon^2 / m$
by (*auto simp: assms*)
also have $(\{::'m \text{ set}\} \neq UNIV \wedge \text{finite } (UNIV :: 'm \text{ set}))$
by *simp*
ultimately have $(\sum_{i \in UNIV}. ?c^2 * ((\| ?u \ x \ i \|^2) < (\sum_{(i::'m) \in UNIV}. \varepsilon^2 / m)$
by (*metis (lifting) sum-strict-mono*)
moreover have $?c^2 * (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2) = (\sum_{i \in UNIV}. ?c^2 * (\| ?u \ x \ i \|^2))$
using *sum-distrib-left by blast*
ultimately have $?c^2 * (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2) < \varepsilon^2$
by (*simp add: assms*)
hence $\text{sqrt } (?c^2 * (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2)) < \text{sqrt } (\varepsilon^2)$
using *real-sqrt-less-iff by blast*
also have $\dots = \varepsilon$
using $\langle 0 < \varepsilon \rangle$ **by** *auto*
moreover have $?c * \text{sqrt } (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2) = \text{sqrt } (?c^2 * (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2))$
by (*simp add: real-sqrt-mult*)
ultimately show $?c * \text{sqrt } (\sum_{i \in UNIV}. (\| ?u \ x \ i \|^2) < \varepsilon$
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 (*rename-tac e z*) (*erule-tac x=z in allE, simp*)

lemma *tendsto-zero-norm-bound*:

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

lemma *frechet-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 (metis norm-nth-le vector-minus-component vector-scaleR-component)
using assms by simp-all

lemma has-derivative-vec-lambda:
  fixes f::real  $\Rightarrow$  ('a::banach) ^ ('n::finite)
  assumes  $\forall i. D (\lambda t. f\ t\ \$\ i) \mapsto (\lambda h. h\ *_R\ f'\ t\ \$\ i)$  (at t within T)
  shows  $D\ f \mapsto (\lambda h. h\ *_R\ f'\ t)$  at t within T
  apply(unfold has-derivative-def, safe)
  apply(force simp: bounded-linear-def bounded-linear-axioms-def)
  using assms frechet-vec-lambda[of - f] unfolding has-derivative-def by auto

lemma has-derivative-vec-nth:
  assumes  $D\ f \mapsto (\lambda h. h\ *_R\ f'\ t)$  at t within T
  shows  $D (\lambda t. f\ t\ \$\ i) \mapsto (\lambda h. h\ *_R\ f'\ t\ \$\ i)$  at t 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

```

0.2 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-prelims-dyn-sys
  imports hs-prelims
begin

```

0.2.1 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 *real set* \Rightarrow '*a set* \Rightarrow
 \Rightarrow
real \Rightarrow '*a* \Rightarrow (*real* \Rightarrow '*a*) *set* (*Sols*)

where *Sols* *f* *T* *S* *t*₀ *s* = {*X* | *X*. (*D* *X* = (λt . *f* *t* (*X* *t*)) on *T*) \wedge *X* *t*₀ = *s* \wedge *X* \in *T* \rightarrow *S*}

lemma *ivp-solsI*:

assumes *D* *X* = (λt . *f* *t* (*X* *t*)) on *T* *X* *t*₀ = *s* *X* \in *T* \rightarrow *S*
shows *X* \in *Sols* *f* *T* *S* *t*₀ *s*
using *assms* **unfolding** *ivp-sols-def* **by** *blast*

lemma *ivp-solsD*:

assumes *X* \in *Sols* *f* *T* *S* *t*₀ *s*
shows *D* *X* = (λt . *f* *t* (*X* *t*)) on *T*
and *X* *t*₀ = *s* **and** *X* \in *T* \rightarrow *S*
using *assms* **unfolding** *ivp-sols-def* **by** *auto*

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

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

lemma *g-orbit-eq*:

fixes *X*::('*a*::*preorder*) \Rightarrow '*b*
shows $\gamma \ X \ G \ T = \{X \ t \mid t. \ t \in T \wedge (\forall \tau \in down \ T \ t. \ G \ (X \ \tau))\}$
unfolding *g-orbit-def* **apply** *safe*
using *le-left-mono* **by** *blast auto*

lemma $\gamma \ X \ (\lambda s. \ True) \ T = \{X \ t \mid t. \ t \in T\}$ **for** *X*::('*a*::*preorder*) \Rightarrow '*b*
unfolding *g-orbit-eq* **by** *simp*

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

lemma *g-orbital-eq*: *g-orbital* *f* *G* *T* *S* *t*₀ *s* =

{*X* *t* | *t* *X*. *t* \in *T* \wedge $\mathcal{P} \ X \ (down \ T \ t) \subseteq \{s. \ G \ s\} \wedge X \in Sols \ (\lambda t. \ f) \ T \ S \ t_0 \ s$ }

unfolding *g-orbital-def* *ivp-sols-def* *g-orbit-eq* *image-le-pred* **by** *auto*

lemma *g-orbital* *f* *G* *T* *S* *t*₀ *s* =

{*X* *t* | *t* *X*. *t* \in *T* \wedge (*D* *X* = (*f* \circ *X*) on *T*) \wedge *X* *t*₀ = *s* \wedge *X* \in *T* \rightarrow *S* \wedge ($\mathcal{P} \ X \ (down \ T \ t) \subseteq \{s. \ G \ s\}$)}

unfolding *g-orbital-eq* *ivp-sols-def* **by** *auto*

lemma *g-orbital* *f* *G* *T* *S* *t*₀ *s* = ($\bigcup \ X \in Sols \ (\lambda t. \ f) \ T \ S \ t_0 \ s. \ \gamma \ X \ G \ T$)

unfolding *g-orbital-def* *ivp-sols-def* *g-orbit-eq* **by** *auto*

lemma *g-orbitalI*:

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

lemma *g-orbitalD*:

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

no-notation *g-orbit* (γ)

0.2.2 Differential Invariants

definition *diff-invariant* :: $('a \Rightarrow \text{bool}) \Rightarrow (('a :: \text{real-normed-vector}) \Rightarrow 'a) \Rightarrow \text{real set} \Rightarrow$

$'a \text{ set} \Rightarrow \text{real} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where *diff-invariant* $I \ f \ T \ S \ t_0 \ G \equiv (\bigcup \circ (\mathcal{P} \ (g\text{-orbital } f \ G \ T \ S \ t_0))) \ \{s. \ I \ s\} \subseteq \{s. \ I \ s\}$

lemma *diff-invariant-eq*: *diff-invariant* $I \ f \ T \ S \ t_0 \ G =$

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

unfolding *diff-invariant-def* *g-orbital-eq* *image-le-pred* **by** *auto*

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

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

unfolding *diff-invariant-eq* *g-orbital-eq* *image-le-pred* **by** *auto*

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

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

assumes *Thyp*: *is-interval* $T \ t_0 \in T$

and $\forall X. (D \ X = (\lambda \tau. f \ (X \ \tau)) \text{ on } T) \longrightarrow (D \ (\lambda \tau. \mu \ (X \ \tau) - \nu \ (X \ \tau)) = ((*_R) \ 0) \text{ on } T)$

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

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

fix $X \ \tau$ **assume** *tHyp*: $\tau \in T$ **and** *x-ivp*: $D \ X = (\lambda \tau. f \ (X \ \tau)) \text{ on } T \ \mu \ (X \ t_0) = \nu \ (X \ t_0)$

hence *obs1*: $\forall t \in T. D \ (\lambda \tau. \mu \ (X \ \tau) - \nu \ (X \ \tau)) \mapsto (\lambda \tau. \tau *_R \ 0) \text{ at } t \text{ within } T$

using *assms* **by** (*auto* *simp*: *has-vderiv-on-def* *has-vector-derivative-def*)

have *obs2*: $\{t_0 \dashv\tau\} \subseteq T$

using *closed-segment-subset-interval* *tHyp* *Thyp* **by** *blast*

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

using *obs1* *x-ivp* **by** (*auto* *intro!*: *has-derivative-subset*[*OF* - *obs2*]

simp: *has-vderiv-on-def* *has-vector-derivative-def*)

then obtain t **where** $t \in \{t_0 \dashv\tau\}$ **and** $\mu \ (X \ \tau) - \nu \ (X \ \tau) - (\mu \ (X \ t_0) - \nu \ (X \ t_0)) = 0$

$(X\ t_0)) = (\tau - t_0) * t *_R 0$
using *mvt-very-simple-closed-segmentE* **by** *blast*
thus $\mu\ (X\ \tau) = \nu\ (X\ \tau)$
by (*simp add: x-ivp(2)*)
qed

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

fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
assumes *Thyp: is-interval* $T\ t_0 \in T$
and $\forall X. (D\ X = (\lambda\tau. f\ (X\ \tau))\ \text{on}\ T) \longrightarrow (\forall \tau \in T. (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
 $(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau. \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ \text{on}\ T)$
shows *diff-invariant* $(\lambda s. \nu\ s \leq \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add: diff-invariant-eq ivp-sols-def, clarsimp*)
fix $X\ \tau$ **assume** $\tau \in T$ **and** *x-ivp*: $D\ X = (\lambda\tau. f\ (X\ \tau))\ \text{on}\ T$ $\nu\ (X\ t_0) \leq \mu\ (X\ t_0)$
{assume $\tau \neq t_0$
hence *primed*: $\bigwedge \tau. \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
 $\bigwedge \tau. \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall t \in T. D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau. \tau *_R (\mu'\ (X\ t) - \nu'\ (X\ t)))\ \text{at}\ t\ \text{within}\ T$
using *assms x-ivp* **by** (*auto simp: has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0 <--< \tau\} \subseteq T\ \{t_0--\tau\} \subseteq T$
using $\langle \tau \in T \rangle$ *Thyp* $\langle \tau \neq t_0 \rangle$ **by** (*auto simp: convex-contains-open-segment is-interval-convex-1 closed-segment-subset-interval*)
hence $D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau. \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ \text{on}\ \{t_0--\tau\}$
using *obs1 x-ivp* **by** (*auto intro!: has-derivative-subset[OF - obs2(2)] simp: has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0 <--< \tau\}$ **and**
 $(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau. \tau * (\mu'\ (X\ t) - \nu'\ (X\ t)))\ (\tau - t_0)$
using *mvt-simple-closed-segmentE* $\langle \tau \neq t_0 \rangle$ **by** *blast*
hence *mvt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
by *force*
have $\tau > t_0 \Longrightarrow t > t_0 \neg t_0 \leq \tau \Longrightarrow t < t_0\ t \in T$
using $\langle t \in \{t_0 <--< \tau\} \rangle$ *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover **have** $t > t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0\ t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \leq 0$
using *primed(1,2)[OF* $\langle t \in T \rangle$ **]** **by** *auto*
ultimately **have** $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$
apply(*case-tac* $\tau \geq t_0$) **by** (*force, auto simp: split-mult-pos-le*)
hence $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) \geq 0$
using *x-ivp(2)* **by** *auto*
hence $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$
using *mvt* **by** *simp*
thus $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$

using *x-ivp* by *blast*
qed

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

fixes $\mu::'a::\text{banach} \Rightarrow \text{real}$
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall X. (D\ X = (\lambda\tau. f\ (X\ \tau)) \text{ on } T) \longrightarrow (\forall \tau \in T. (\tau > t_0 \longrightarrow \mu' (X\ \tau) \geq \nu' (X\ \tau)) \wedge (\tau < t_0 \longrightarrow \mu' (X\ \tau) \leq \nu' (X\ \tau))) \wedge (D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau. \mu' (X\ \tau) - \nu' (X\ \tau)) \text{ on } T)$
shows *diff-invariant* $(\lambda s. \nu\ s < \mu\ s) f\ T\ S\ t_0\ G$
proof(*simp add: diff-invariant-eq ivp-sols-def, clarsimp*)
fix $X\ \tau$ **assume** $\tau \in T$ **and** *x-ivp*: $D\ X = (\lambda\tau. f\ (X\ \tau)) \text{ on } T$ $\nu\ (X\ t_0) < \mu\ (X\ t_0)$
{assume $\tau \neq t_0$
hence *primed*: $\bigwedge \tau. \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu' (X\ \tau) \geq \nu' (X\ \tau)$
 $\bigwedge \tau. \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu' (X\ \tau) \leq \nu' (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall t \in T. D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau. \tau *_R (\mu' (X\ t) - \nu' (X\ t))) \text{ at } t \text{ within } T$
using *assms x-ivp* **by** (*auto simp: has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0 < \tau < t_0\} \subseteq T \subseteq \{t_0 < \tau < t_0\} \subseteq T$
using $\langle \tau \in T \rangle$ *Thyp* $\langle \tau \neq t_0 \rangle$ **by** (*auto simp: convex-contains-open-segment is-interval-convex-1 closed-segment-subset-interval*)
hence $D\ (\lambda\tau. \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau. \mu' (X\ \tau) - \nu' (X\ \tau)) \text{ on } \{t_0 < \tau < t_0\}$
using *obs1 x-ivp* **by** (*auto intro!: has-derivative-subset[OF - obs2(2)] simp: has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0 < \tau < t_0\}$ **and**
 $(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau. \tau * (\mu' (X\ t) - \nu' (X\ t))) (\tau - t_0)$
using *mt-simple-closed-segmentE* $\langle \tau \neq t_0 \rangle$ **by** *blast*
hence *mt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu' (X\ t) - \nu' (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
by *force*
have $\tau > t_0 \Longrightarrow t > t_0 \wedge t_0 \leq \tau \Longrightarrow t < t_0\ t \in T$
using $\langle t \in \{t_0 < \tau < t_0\} \rangle$ *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover **have** $t > t_0 \Longrightarrow (\mu' (X\ t) - \nu' (X\ t)) \geq 0\ t < t_0 \Longrightarrow (\mu' (X\ t) - \nu' (X\ t)) \leq 0$
using *primed*(1,2)[*OF* $\langle t \in T \rangle$] **by** *auto*
ultimately **have** $(\tau - t_0) * (\mu' (X\ t) - \nu' (X\ t)) \geq 0$
apply(*case-tac* $\tau \geq t_0$) **by** (*force, auto simp: split-mult-pos-le*)
hence $(\tau - t_0) * (\mu' (X\ t) - \nu' (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) > 0$
using *x-ivp*(2) **by** *auto*
hence $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
using *mt* **by** *simp*
thus $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
using *x-ivp* **by** *blast*
qed

lemma *diff-invariant-conj-rule* [*diff-invariant-rules*]:
assumes *diff-invariant* $I_1 f T S t_0 G$
and *diff-invariant* $I_2 f T S t_0 G$
shows *diff-invariant* $(\lambda s. I_1 s \wedge I_2 s) f T S t_0 G$
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

lemma *diff-invariant-disj-rule* [*diff-invariant-rules*]:
assumes *diff-invariant* $I_1 f T S t_0 G$
and *diff-invariant* $I_2 f T S t_0 G$
shows *diff-invariant* $(\lambda s. I_1 s \vee I_2 s) f T S t_0 G$
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

0.2.3 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 $f :: \text{real} \Rightarrow ('a :: \{\text{heine-borel}, \text{banach}\}) \Rightarrow 'a$ **and** $T :: \text{real set}$ **and** $S :: 'a \text{ set}$
and $t_0 :: \text{real}$
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*)

lemmas *subintervalI* = *closed-segment-subset-domain*

lemma *csols-eq*: $\text{csols } t_0 s = \{(X, t). t \in T \wedge X \in \text{Sols } f \{t_0 \dots t\} S t_0 s\}$
unfolding *ivp-sols-def csols-def solves-ode-def* **using** *subintervalI* [*OF init-time*]
by *auto*

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

lemma *unique-solution*:
assumes *xivp*: $D X = (\lambda t. f t (X t))$ *on* $\{t_0 \dots t\}$ $X t_0 = s$ $X \in \{t_0 \dots t\} \rightarrow S$
and $t \in T$
and *yivp*: $D Y = (\lambda t. f t (Y t))$ *on* $\{t_0 \dots t\}$ $Y t_0 = s$ $Y \in \{t_0 \dots t\} \rightarrow S$ **and**
 $s \in S$
shows $X t = Y t$
proof–
have $(X, t) \in \text{csols } t_0 s$
using *xivp* $\langle t \in T \rangle$ **unfolding** *csols-eq ivp-sols-def* **by** *auto*
hence *ivl-fact*: $\{t_0 \dots t\} \subseteq \text{ex-ivl } s$
unfolding *existence-ivl-def* **by** *auto*

have $obs: \bigwedge z T'. t_0 \in T' \wedge is\text{-interval } T' \wedge T' \subseteq ex\text{-ivl } s \wedge (z \text{ solves-ode } f) T'$
 $S \implies$
 $z t_0 = flow\ t_0\ s\ t_0 \implies (\forall t \in T'. z\ t = flow\ t_0\ s\ t)$
using $flow\text{-usolves-ode}[OF\ init\text{-time } \langle s \in S \rangle]$ **unfolding** $usolves\text{-ode-from-def}$
by $blast$
have $\forall \tau \in \{t_0 \dashv\!\!\dashv\!\! t\}. X\ \tau = flow\ t_0\ s\ \tau$
using $obs[of\ \{t_0 \dashv\!\!\dashv\!\! t\}\ X]$ $xivp\ ivl\text{-fact}\ flow\text{-initial-time}[OF\ init\text{-time } \langle s \in S \rangle]$
unfolding $solves\text{-ode-def}$ **by** $simp$
also have $\forall \tau \in \{t_0 \dashv\!\!\dashv\!\! t\}. Y\ \tau = flow\ t_0\ s\ \tau$
using $obs[of\ \{t_0 \dashv\!\!\dashv\!\! t\}\ Y]$ $yivp\ ivl\text{-fact}\ flow\text{-initial-time}[OF\ init\text{-time } \langle s \in S \rangle]$
unfolding $solves\text{-ode-def}$ **by** $simp$
ultimately show $X\ t = Y\ t$
by $auto$
qed

lemma $solution\text{-eq-flow}$:

assumes $xivp: D\ X = (\lambda t. f\ t\ (X\ t))$ *on* $ex\text{-ivl } s$ $X\ t_0 = s$ $X \in ex\text{-ivl } s \rightarrow S$
and $t \in ex\text{-ivl } s$ **and** $s \in S$
shows $X\ t = flow\ t_0\ s\ t$
proof–
have $obs: \bigwedge z T'. t_0 \in T' \wedge is\text{-interval } T' \wedge T' \subseteq ex\text{-ivl } s \wedge (z \text{ solves-ode } f) T'$
 $S \implies$
 $z t_0 = flow\ t_0\ s\ t_0 \implies (\forall t \in T'. z\ t = flow\ t_0\ s\ t)$
using $flow\text{-usolves-ode}[OF\ init\text{-time } \langle s \in S \rangle]$ **unfolding** $usolves\text{-ode-from-def}$
by $blast$
have $\forall \tau \in ex\text{-ivl } s. X\ \tau = flow\ t_0\ s\ \tau$
using $obs[of\ ex\text{-ivl } s\ X]$ $existence\text{-ivl-initial-time}[OF\ init\text{-time } \langle s \in S \rangle]$
 $xivp\ flow\text{-initial-time}[OF\ init\text{-time } \langle s \in S \rangle]$ **unfolding** $solves\text{-ode-def}$ **by** $simp$
thus $X\ t = flow\ t_0\ s\ t$
by $(auto\ simp: \langle t \in ex\text{-ivl } s \rangle)$
qed

end

lemma $local\text{-lipschitz-add}$:

fixes $f1\ f2 :: real \Rightarrow 'a :: banach \Rightarrow 'a$
assumes $local\text{-lipschitz } T\ S\ f1$
and $local\text{-lipschitz } T\ S\ f2$
shows $local\text{-lipschitz } T\ S\ (\lambda t\ s. f1\ t\ s + f2\ t\ s)$
proof $(unfold\ local\text{-lipschitz-def, clarsimp)$
fix s **and** t **assume** $s \in S$ **and** $t \in T$
obtain $\varepsilon_1\ L1$ **where** $\varepsilon_1 > 0$ **and** $L1: \bigwedge \tau. \tau \in cball\ t\ \varepsilon_1 \cap T \implies L1\text{-lipschitz-on}$
 $(cball\ s\ \varepsilon_1 \cap S)\ (f1\ \tau)$
using $local\text{-lipschitzE}[OF\ assms(1)\ \langle t \in T \rangle \langle s \in S \rangle]$ **by** $blast$
obtain $\varepsilon_2\ L2$ **where** $\varepsilon_2 > 0$ **and** $L2: \bigwedge \tau. \tau \in cball\ t\ \varepsilon_2 \cap T \implies L2\text{-lipschitz-on}$
 $(cball\ s\ \varepsilon_2 \cap S)\ (f2\ \tau)$
using $local\text{-lipschitzE}[OF\ assms(2)\ \langle t \in T \rangle \langle s \in S \rangle]$ **by** $blast$
have $ballH: cball\ s\ (\min\ \varepsilon_1\ \varepsilon_2) \cap S \subseteq cball\ s\ \varepsilon_1 \cap S \cap cball\ s\ (\min\ \varepsilon_1\ \varepsilon_2) \cap S \subseteq$
 $cball\ s\ \varepsilon_2 \cap S$

by *auto*
have *obs1*: $\forall \tau \in \text{cball } t \ \varepsilon_1 \cap T. \text{L1-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S) \ (f1 \ \tau)$
using *lipschitz-on-subset*[*OF L1 ballH(1)*] **by** *blast*
also have *obs2*: $\forall \tau \in \text{cball } t \ \varepsilon_2 \cap T. \text{L2-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S)$
(f2 τ)
using *lipschitz-on-subset*[*OF L2 ballH(2)*] **by** *blast*
ultimately have $\forall \tau \in \text{cball } t \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap T.$
 $(\text{L1} + \text{L2})\text{-lipschitz-on } (\text{cball } s \ (\min \ \varepsilon_1 \ \varepsilon_2) \cap S) \ (\lambda s. f1 \ \tau \ s + f2 \ \tau \ s)$
using *lipschitz-on-add* **by** *fastforce*
thus $\exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap T. \text{L-lipschitz-on } (\text{cball } s \ u \cap S) \ (\lambda s. f1 \ t \ s +$
 $f2 \ t \ s)$
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*: $\text{picard-lindelof } f1 \ T \ S \ t_0 \implies \text{picard-lindelof } f2 \ T \ S \ t_0 \implies$
 $\text{picard-lindelof } (\lambda t \ s. f1 \ t \ s + f2 \ t \ s) \ T \ S \ t_0$
unfolding *picard-lindelof-def* **apply**(*clarsimp*, *rule conjI*)
using *continuous-on-add* **apply** *fastforce*
using *local-lipschitz-add* **by** *blast*

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

0.2.4 Flows for ODEs

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

locale *local-flow* = *picard-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*:
 $\bigwedge t \ s. t \in T \implies s \in S \implies D \ (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } \{0 \dashv\dashv t\}$
 $\bigwedge s. s \in S \implies \varphi \ 0 \ s = s$
 $\bigwedge t \ s. t \in T \implies s \in S \implies (\lambda t. \varphi \ t \ s) \in \{0 \dashv\dashv t\} \rightarrow S$
begin

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

lemma *eq-solution-ivl*:
assumes *xivp*: $D \ X = (\lambda t. f \ (X \ t)) \text{ on } \{0 \dashv\dashv t\} \ X \ 0 = s \ X \in \{0 \dashv\dashv t\} \rightarrow S$
and *indom*: $t \in T \ s \in S$

shows $X\ t = \varphi\ t\ s$
apply(*rule unique-solution*[*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 $ex\text{-}ivl\ s = T$
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 $D\ (\lambda\tau. \varphi\ \tau\ s) \mapsto (\lambda\tau. \tau *_{\mathbb{R}} f\ (\varphi\ t\ s))$ *at* t *within* B
proof–
obtain $r::real$ **where** *rHyp*: $r > 0\ ball\ t\ r \subseteq T$
using *open-contains-ball-eq open-domain*(1) $\langle t \in T \rangle$ **by** *blast*
moreover **have** $t + r/2 > 0$
using $\langle r > 0 \rangle\ \langle t > 0 \rangle$ **by** *auto*
moreover **have** $\{0--t\} \subseteq T$
using *subintervalI*[*OF init-time* $\langle t \in T \rangle$] .
ultimately **have** *subs*: $\{0<--<t + r/2\} \subseteq T$
unfolding *abs-le-eq abs-le-eq real-ivl-eqs*[*OF* $\langle t > 0 \rangle$] *real-ivl-eqs*[*OF* $\langle t + r/2 > 0 \rangle$]
by *clarify* (*case-tac* $t < x$, *simp-all add: cball-def ball-def dist-norm subset-eq field-simps*)
have $t + r/2 \in T$
using *rHyp* **unfolding** *real-ivl-eqs*[*OF rHyp*(1)] **by** (*simp add: subset-eq*)
hence $\{0--t + r/2\} \subseteq T$
using *subintervalI*[*OF init-time*] **by** *blast*
hence $(D\ (\lambda t. \varphi\ t\ s) = (\lambda t. f\ (\varphi\ t\ s)))$ *on* $\{0--(t + r/2)\}$
using *ivp*(1)[*OF -* $\langle s \in S \rangle$] **by** *auto*
hence *vderiv*: $(D\ (\lambda t. \varphi\ t\ s) = (\lambda t. f\ (\varphi\ t\ s)))$ *on* $\{0<--<t + r/2\}$
apply(*rule has-vderiv-on-subset*)
unfolding *real-ivl-eqs*[*OF* $\langle t + r/2 > 0 \rangle$] **by** *auto*
have $t \in \{0<--<t + r/2\}$
unfolding *real-ivl-eqs*[*OF* $\langle t + r/2 > 0 \rangle$] **using** *rHyp* $\langle t > 0 \rangle$ **by** *simp*
moreover **have** $D\ (\lambda\tau. \varphi\ \tau\ s) \mapsto (\lambda\tau. \tau *_{\mathbb{R}} f\ (\varphi\ t\ s))$ (*at* t *within* $\{0<--<t + r/2\}$)
using *vderiv calculation* **unfolding** *has-vderiv-on-def has-vector-derivative-def*
by *blast*
moreover **have** *open* $\{0<--<t + r/2\}$
unfolding *real-ivl-eqs*[*OF* $\langle t + r/2 > 0 \rangle$] **by** *simp*
ultimately **show** *?thesis*
using *subs that* **by** *blast*
qed

lemma *has-derivative-on-open2*:

assumes $t < 0 \ t \in T \ s \in S$
 obtains B where $t \in B$ and open B and $B \subseteq T$
 and $D \ (\lambda\tau. \varphi \ \tau \ s) \mapsto (\lambda\tau. \tau *_R f \ (\varphi \ t \ s))$ at t within B
 proof–
 obtain $r::real$ 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 open-segment-eq-real-ivl closed-segment-eq-real-ivl
 real-ivl-eqs[OF rHyp(1)] by(auto simp: subset-eq)
 have $t - r/2 \in T$
 using rHyp unfolding real-ivl-eqs by (simp add: subset-eq)
 hence $\{0--t - r/2\} \subseteq T$
 using subintervalI[OF init-time] by blast
 hence $(D \ (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$ on $\{0--(t - r/2)\}$
 using ivp(1)[OF - $\langle s \in S \rangle$] by auto
 hence vderiv: $(D \ (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)))$ on $\{0<--<t - r/2\}$
 apply(rule has-vderiv-on-subset)
 unfolding open-segment-eq-real-ivl closed-segment-eq-real-ivl by auto
 have $t \in \{0<--<t - r/2\}$
 unfolding open-segment-eq-real-ivl using rHyp $\langle t < 0 \rangle$ by simp
 moreover have $D \ (\lambda\tau. \varphi \ \tau \ s) \mapsto (\lambda\tau. \tau *_R f \ (\varphi \ t \ s))$ (at t within $\{0<--<t - r/2\}$)
 using vderiv calculation unfolding has-vderiv-on-def has-vector-derivative-def
 by blast
 moreover have open $\{0<--<t - r/2\}$
 unfolding 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 $r::real$ 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: $\{0--r/2\} \subseteq T \ \{0--(-r/2)\} \subseteq T$
 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*:
 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } \{0--r/2\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\})$
 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } \{0--(-r/2)\} \cup \text{closure } \{0--r/2\} \cap \text{closure } \{0--(-r/2)\})$
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 $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } \{-r/2--r/2\})$
using *has-vderiv-on-union*[*OF vderivs*] **by** *simp*
hence $(D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } \{-r/2 <--< r/2\})$
using *has-vderiv-on-subset*[*OF - segment-open-subset-closed*[*of -r/2 r/2*]] **by** *auto*
hence $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ 0 \ s)) \text{ (at } 0 \text{ within } \{-r/2 <--< r/2\})$
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 $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ t \ s)) \text{ at } t \text{ within } B$
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 $(\lambda t. \varphi \ t \ s) \in T \rightarrow S$
unfolding *ex-ivl-eq*[*symmetric*] *existence-ivl-def*
using *local.mem-existence-ivl-subset* *ivp*(3)[*OF - assms*] **by** *blast*

lemma *has-vderiv-on-domain*:

assumes $s \in S$
shows $D (\lambda t. \varphi \ t \ s) = (\lambda t. f \ (\varphi \ t \ s)) \text{ on } T$
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*: $D (\lambda \tau. \varphi \ \tau \ s) \mapsto (\lambda \tau. \tau *_R f \ (\varphi \ t \ s)) \text{ at } t \text{ within } B$
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 $D (\lambda t. \varphi \ t \ s) \mapsto (\lambda \tau. \tau *_{\mathcal{R}} f (\varphi \ t \ s))$ 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 $s \in S$
 shows $(\lambda t. \varphi \ t \ s) \in \text{Sols } (\lambda t. f) \ T \ S \ 0 \ s$
 using *has-vderiv-on-domain ivp*(2) *in-domain* apply(*rule ivp-solsI*)
 using *assms* by *auto*

lemma *eq-solution*:
 assumes $X \in \text{Sols } (\lambda t. f) \ T \ S \ 0 \ s$ and $t \in T$ and $s \in S$
 shows $X \ t = \varphi \ t \ s$
 proof—
 have $D \ X = (\lambda t. f \ (X \ t))$ on $(\text{ex-ivl } s)$ and $X \ 0 = s$ and $X \in (\text{ex-ivl } s) \rightarrow S$
 using *ivp-solsD*[*OF* *assms*(1)] *unfolding* *ex-ivl-eq*[*OF* $\langle s \in S \rangle$] by *auto*
 note *solution-eq-flow*[*OF* *this*]
 hence $X \ t = \text{flow } 0 \ s \ t$
 unfolding *ex-ivl-eq*[*OF* $\langle s \in S \rangle$] using *assms* by *blast*
 also have $\varphi \ t \ s = \text{flow } 0 \ s \ t$
 apply(*rule solution-eq-flow ivp*)
 apply(*simp-all* add: *assms*(2,3) *ivp*(2)[*OF* $\langle s \in S \rangle$])
 unfolding *ex-ivl-eq*[*OF* $\langle s \in S \rangle$] by (*auto simp: has-vderiv-on-domain assms*
in-domain)
 ultimately show $X \ t = \varphi \ t \ s$
 by *simp*
 qed

lemma *ivp-sols-collapse*:
 assumes $T = \text{UNIV}$ and $s \in S$
 shows $\text{Sols } (\lambda t. f) \ T \ S \ 0 \ s = \{(\lambda t. \varphi \ t \ s)\}$
 using *in-ivp-sols eq-solution* *assms* by *auto*

lemma *additive-in-ivp-sols*:
 assumes $s \in S$ and $\mathcal{P} (\lambda \tau. \tau + t) \ T \subseteq T$
 shows $(\lambda \tau. \varphi \ (\tau + t) \ s) \in \text{Sols } (\lambda t. f) \ T \ S \ 0 \ (\varphi \ (0 + t) \ s)$
 apply(*rule ivp-solsI*, *rule vderiv-on-compose-intro*[*OF* *has-vderiv-on-subset*])
 apply(*rule has-vderiv-on-domain*)
 using *in-domain* *assms* by (*auto intro: derivative-intros*)

lemma *is-monoid-action*:
 assumes $s \in S$ and $T = \text{UNIV}$
 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  $\varphi \ t_2 \ s \in S$ 
    using in-domain assms by auto
  finally 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

```

```

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

```

```

lemma orbit-eq[simp]:
  assumes  $s \in S$ 
  shows  $\gamma^\varphi \ s = \{\varphi \ t \ s \mid t. t \in T\}$ 
  using eq-solution assms unfolding orbit-def g-orbital-eq ivp-sols-def
  by(auto intro!: has-vderiv-on-domain ivp(2) in-domain)

```

```

lemma g-orbital-collapses:
  assumes  $s \in S$ 
  shows  $g\text{-orbital } f \ G \ T \ S \ 0 \ s = \{\varphi \ t \ s \mid t. t \in T \wedge (\forall \tau \in \text{down } T \ t. G \ (\varphi \ \tau \ s))\}$ 
proof(rule subset-antisym, simp-all only: subset-eq)
  let ?gorbit =  $\{\varphi \ t \ s \mid t. t \in T \wedge (\forall \tau \in \text{down } T \ t. G \ (\varphi \ \tau \ s))\}$ 
  {fix  $s'$  assume  $s' \in g\text{-orbital } f \ G \ T \ S \ 0 \ s$ 
    then obtain  $X$  and  $t$  where  $x\text{-ivp}: X \in \text{Sols } (\lambda t. f) \ T \ S \ 0 \ s$ 
      and  $X \ t = s'$  and  $t \in T$  and  $\text{guard}:(\mathcal{P} \ X \ (\text{down } T \ t) \subseteq \{s. G \ s\})$ 
      unfolding g-orbital-def g-orbit-eq by auto
    have  $\text{obs}:\forall \tau \in (\text{down } T \ t). X \ \tau = \varphi \ \tau \ s$ 
      using eq-solution[OF x-ivp - assms] by blast
    hence  $\mathcal{P} \ (\lambda t. \varphi \ t \ s) \ (\text{down } T \ t) \subseteq \{s. G \ s\}$ 
      using guard by auto
    also have  $\varphi \ t \ s = X \ t$ 
      using eq-solution[OF x-ivp  $\langle t \in T \rangle$  assms] by simp
    ultimately have  $s' \in ?gorbit$ 
      using  $\langle X \ t = s' \rangle \langle t \in T \rangle$  by auto}
  thus  $\forall s' \in g\text{-orbital } f \ G \ T \ S \ 0 \ s. s' \in ?gorbit$ 
    by blast
next
  let ?gorbit =  $\{\varphi \ t \ s \mid t. t \in T \wedge (\forall \tau \in \text{down } T \ t. G \ (\varphi \ \tau \ s))\}$ 
  {fix  $s'$  assume  $s' \in ?gorbit$ 
    then obtain  $t$  where  $\mathcal{P} \ (\lambda t. \varphi \ t \ s) \ (\text{down } T \ t) \subseteq \{s. G \ s\}$  and  $t \in T$  and  $\varphi \ t \ s = s'$ 
      by blast
    hence  $s' \in g\text{-orbital } f \ G \ T \ S \ 0 \ s$ 
      using assms by(auto intro!: g-orbitalI in-ivp-sols)}
  thus  $\forall s' \in ?gorbit. s' \in g\text{-orbital } f \ G \ T \ S \ 0 \ s$ 
    by blast
qed

```

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 derivative-intros(191))
  apply(rule derivative-intros, simp)+
  by simp-all

```

end

0.3 Linear algebra for hybrid systems

Linear systems of ordinary differential equations (ODEs) are those whose vector fields are linear operators. Broadly speaking, if there is a matrix A such that the system $x't = f(xt)$ can be rewritten as $x't = (At) \cdot (xt) + (Bt)$, then the system is called linear. The end goal of this section is to prove that every linear system of ODEs has a unique solution, and to obtain a characterization of said solution. We start by formalising various properties of vector spaces.

theory *hs-prelims-matrices*

imports

hs-prelims-dyn-sys

Affine-Arithmetic.Executable-Euclidean-Space

begin

0.3.1 Properties of some vector operations

abbreviation $e\ k \equiv axis\ k\ 1$

notation *matrix-inv* $(^{-1}\ [90])$

abbreviation *entries* $(A::'a^{n \times m}) \equiv \{A\ \$\ i\ \$\ j \mid i\ j. i \in UNIV \wedge j \in UNIV\}$

lemma *finite-sum-univ-singleton*: $(sum\ g\ UNIV) = sum\ g\ \{i\} + sum\ g\ (UNIV - \{i\})$ **for** $i :: 'a::finite$

by $(metis\ add.commute\ finite-class.finite-UNIV\ sum.subset-diff\ top-greatest)$

lemma *kronecker-delta-simps*[*simp*]:

fixes $q :: ('a::semiring-0)$ **and** $i :: 'n::finite$

shows $(\sum_{j \in UNIV}. f\ j * (if\ j = i\ then\ q\ else\ 0)) = f\ i * q$

and $(\sum_{j \in UNIV}. f\ j * (if\ i = j\ then\ q\ else\ 0)) = f\ i * q$

and $(\sum_{j \in UNIV}. (if\ i = j\ then\ q\ else\ 0) * f\ j) = q * f\ i$

and $(\sum_{j \in UNIV}. (if\ j = i\ then\ q\ else\ 0) * f\ j) = q * f\ i$

by $(auto\ simp: finite-sum-univ-singleton[of\ -\ i])$

lemma *sum-axis*[simp]:
 fixes $q :: ('a::semiring-0)$
 shows $(\sum_{j \in UNIV}. f\ j * axis\ i\ q\ \$\ j) = f\ i * q$
 and $(\sum_{j \in UNIV}. axis\ i\ q\ \$\ j * f\ j) = q * f\ i$
 unfolding *axis-def* by(auto simp: vec-eq-iff)

lemma *sum-scalar-nth-axis*: $sum\ (\lambda i. (x\ \$\ i) * s\ e\ i)\ UNIV = x$ for $x :: ('a::semiring-1)^n$
 unfolding *vec-eq-iff* *axis-def* by simp

lemma *scalar-eq-scaleR*[simp]: $c * s\ x = c *_R\ x$
 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*: $(A * v\ v) \cdot w = v \cdot (transpose\ A * v\ w)$ for $A :: real^n^n$
 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]: $- axis\ i\ k = axis\ i\ (-k)$ for $k :: 'a::ring$
 unfolding *axis-def* by(simp add: vec-eq-iff)

lemma *norm-axis-eq*[simp]: $\|axis\ i\ k\| = \|k\|$
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 $(\sum_{j \in UNIV}. (\|?\delta_K\ j\ i\ k\|\|^2)) = (\sum_{j \in \{i\}}. (\|?\delta_K\ j\ i\ k\|\|^2)) + (\sum_{j \in (UNIV - \{i\})}. (\|?\delta_K\ j\ i\ k\|\|^2))$
 using *finite-sum-univ-singleton* by blast
 also have $\dots = (\|k\|\|^2)$ by simp
 finally show $\sqrt{(\sum_{j \in UNIV}. (norm\ (\text{if } j = i \text{ then } k \text{ else } 0))^2)} = norm\ k$ by
 simp
 qed

lemma *matrix-axis-0*:
 fixes $A :: ('a::idom)^n^m$
 assumes $k \neq 0$ and $h:\forall i. (A * v\ (axis\ i\ k)) = 0$
 shows $A = 0$
proof–
 {fix $i :: 'n$
 have $0 = (\sum_{j \in UNIV}. (axis\ i\ k)\ \$\ j * s\ column\ j\ A)$
 using *h* *matrix-mult-sum*[of $A\ axis\ i\ k$] by simp
 also have $\dots = k * s\ column\ i\ A$
 by(simp add: *axis-def* *vector-scalar-mult-def* *column-def* *vec-eq-iff* *mult commute*)
 finally have $k * s\ column\ i\ A = 0$
 unfolding *axis-def* by simp
 hence $column\ i\ A = 0$

```

    using vector-mul-eq-0 (k ≠ 0) by blast}
  thus A = 0
    unfolding column-def vec-eq-iff by simp
qed

```

```

lemma scaleR-norm-sgn-eq: (||x||) *R 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::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::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::real-normed-algebra-1) ^'n
  shows (||A *v sgn x||) * (||x||) = ||A *v x||
proof-
  have ||A *v x|| = ||A *v ((||x||) *R sgn x)||
    by (simp add: scaleR-norm-sgn-eq)
  also have ... = (||A *v sgn x||) * (||x||)
    by (simp add: vector-scaleR-commute)
  finally show ?thesis ..
qed

```

0.3.2 Matrix norms

Here we develop the foundations for obtaining the Lipschitz constant for every system of ODEs of the form $x' t = A * v x t$. We derive some properties of two matrix norms.

Matrix operator norm

```

abbreviation op-norm :: ('a::real-normed-algebra-1) ^'n ^'m ⇒ real ((1 ||-||op) [65]
61)
  where ||A||op ≡ onorm (λx. A *v x)

```

```

lemma norm-matrix-bound:
  fixes A :: ('a::real-normed-algebra-1) ^'n ^'m
  shows ||x|| = 1 ⇒ ||A *v x|| ≤ ||(χ i j. ||A $ i $ j||) *v 1||
proof-
  fix x :: ('a, 'n) vec assume ||x|| = 1
  hence xi-le1: ⋀i. ||x $ i|| ≤ 1
    by (metis Finite-Cartesian-Product.norm-nth-le)
  {fix j::'m
   have ||(∑ i∈UNIV. A $ j $ i * x $ i)|| ≤ (∑ i∈UNIV. ||A $ j $ i * x $ i||)
     using norm-sum by blast
   also have ... ≤ (∑ i∈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-le1* by (*simp* add: *sum-mono mult-left-le*)
 finally have $\|(\sum_{i \in UNIV}. A \$ j \$ i * x \$ i)\| \leq (\sum_{i \in UNIV}. (\|A \$ j \$ i\|$
 $* 1) \text{ by } \textit{simp}\}$
 hence $\bigwedge j. \|A * v x \$ j\| \leq ((\chi \text{ i1 i2}. \|A \$ i1 \$ i2\|) * v 1) \$ j$
 unfolding *matrix-vector-mult-def* by *simp*
 hence $(\sum_{j \in UNIV}. (\|(A * v x) \$ j\|)^2) \leq (\sum_{j \in UNIV}. (((\chi \text{ i1 i2}. \|A \$ i1 \$$
 $i2\|) * v 1) \$ j\|)^2)$
 by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def*
sum-mono)
 thus $\|A * v x\| \leq \|(\chi \text{ i j}. \|A \$ i \$ j\|) * v 1\|$
 unfolding *norm-vec-def L2-set-def* by *simp*
 qed

lemma *onorm-set-proptys*:

fixes $A :: ('a::\text{real-normed-algebra-1})^n^m$
 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*)
 apply(*rule-tac* $x=\|(\chi \text{ i j}. \|A \$ i \$ j\|) * v 1\|$ in *exI*, *clarsimp*,
subst mult-norm-matrix-sgn-eq[symmetric], *clarsimp*,
rule-tac $x=\text{sgn} -$ in *norm-matrix-bound*, *simp* add: *norm-sgn*)+
 by *force*

lemma *op-norm-set-proptys*:

fixes $A :: ('a::\text{real-normed-algebra-1})^n^m$
 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* $x=\|(\chi \text{ i j}. \|A \$ i \$ j\|) * v 1\|$ in *exI*)
 apply(*force simp: norm-matrix-bound dist-real-def*)
 apply(*rule-tac* $x=\|(\chi \text{ i j}. \|A \$ i \$ j\|) * v 1\|$ in *exI*, *force simp: norm-matrix-bound*)
 using *ex-norm-eq-1* by *blast*

lemma *op-norm-def*:

fixes $A :: ('a::\text{real-normed-algebra-1})^n^m$
 shows $\|A\|_{op} = \text{Sup } \{\|A * v x\| \mid x. \|x\| = 1\}$
 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-zero-iff*: $(\|A\|_{op} = 0) = (A = 0)$ **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-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*:

fixes $A :: ('a::\text{real-normed-algebra-1})^{n \times m}$

shows $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$

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*] $\langle 0 \leq (\|A\|_{op}) \rangle$ *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** $A :: \text{real}^{n' n}$

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 $\dots \leq (\|x\|) * (\|transpose\ A * v\ (A * v\ x)\|)$
using *norm-cauchy-schwarz* **by** *blast*
also have $\dots \leq (\|transpose\ A ** A\|_{op}) * (\|x\|)^2$
apply(*subst matrix-vector-mul-assoc*)
using *norm-matrix-le-mult-op-norm*[*of* $transpose\ A ** A\ x$]
by (*simp add*: $\langle \|x\| = 1 \rangle$)
finally have $(\|A * v\ x\|)^2 \leq (\|transpose\ A ** A\|_{op}) * (\|x\|)^2$
by *linarith*
thus $\|A * v\ x\| \leq \text{sqrt} ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
by (*simp add*: $\langle \|x\| = 1 \rangle$ *real-le-rsqrt*)
qed

lemma *op-norm-le-sum-column*: $\|A\|_{op} \leq (\sum_{i \in UNIV}. \|column\ i\ A\|)$ **for** $A :: \text{real}^{n' n}$

proof(*unfold op-norm-def*, *rule cSup-least*[*OF op-norm-set-proptys*(\mathcal{J})], *clarsimp*)

fix $x :: \text{real}^{n' n}$ **assume** $x\text{-def}:\|x\| = 1$
hence $x\text{-hyp}:\bigwedge i. \|x\ \$\ i\| \leq 1$
by (*simp add*: *norm-bound-component-le-cart*)
have $(\|A * v\ x\|) = \|(\sum_{i \in UNIV}. x\ \$\ i\ * column\ i\ A)\|$
by(*subst matrix-mult-sum*[*of* A], *simp*)
also have $\dots \leq (\sum_{i \in UNIV}. \|x\ \$\ i\ * column\ i\ A\|)$
by (*simp add*: *sum-norm-le*)
also have $\dots = (\sum_{i \in UNIV}. (\|x\ \$\ i\|) * (\|column\ i\ A\|))$
by (*simp add*: *mult-norm-matrix-sgn-eq*)
also have $\dots \leq (\sum_{i \in UNIV}. \|column\ i\ A\|)$
using $x\text{-hyp}$ **by** (*simp add*: *mult-left-le-one-le sum-mono*)
finally show $\|A * v\ x\| \leq (\sum_{i \in UNIV}. \|column\ i\ A\|)$.

qed

lemma *op-norm-le-transpose*: $\|A\|_{op} \leq \|transpose\ A\|_{op}$ **for** $A :: \text{real}^{n' n}$

proof–

have $obs:\forall x. \|x\| = 1 \longrightarrow (\|A * v\ x\|) \leq \text{sqrt} ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
using *norm-matrix-vec-mult-le-transpose* **by** *blast*
have $(\|A\|_{op}) \leq \text{sqrt} ((\|transpose\ A ** A\|_{op}))$
using obs **apply**(*unfold op-norm-def*)
by (*rule cSup-least*[*OF op-norm-set-proptys*(\mathcal{J})]) *clarsimp*
hence $(\|A\|_{op})^2 \leq (\|transpose\ A ** A\|_{op})$

using *power-mono*[of $(\|A\|_{op}) - 2$] *op-norm-ge-0* by *force*
 also have $\dots \leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$
 using *op-norm-matrix-matrix-mult-le* by *blast*
 finally have $((\|A\|_{op}))^2 \leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$ by *linarith*
 thus $(\|A\|_{op}) \leq (\|transpose\ A\|_{op})$
 using *sq-le-cancel*[of $(\|A\|_{op})$] *op-norm-ge-0* by *metis*
 qed

Matrix maximum norm

abbreviation *max-norm* $(A::real^{n \times m}) \equiv Max\ (abs\ ' (entries\ A))$

notation *max-norm* $((1\|-\|_{max})\ [65]\ 61)$

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 $- \cdot Max$], *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 $\bigwedge i. finite\ \{ |A\ \$\ i\ \$\ j| \mid j. j \in UNIV \}$
 using *finite-Atleast-Atmost-nat* by *fastforce*
 hence *finite* $(\bigcup i \in UNIV. \{ |A\ \$\ i\ \$\ j| \mid j. j \in UNIV \})$ (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*: $0 \leq \|A\|_{max}$

proof–

have $\bigwedge i.j. |A\ \$\ i\ \$\ j| \geq 0$ by *simp*
 also have $\bigwedge i.j. |A\ \$\ i\ \$\ j| \leq \|A\|_{max}$
 unfolding *max-norm-def* using *max-norm-set-proptys* *Max-ge* *max-norm-def*
 by *blast*
 finally show $0 \leq \|A\|_{max}$.

qed

lemma *op-norm-le-max-norm*:

fixes $A :: real^{(n::finite) \times (m::finite)}$
 shows $\|A\|_{op} \leq real\ CARD(m) * real\ CARD(n) * (\|A\|_{max})$
 apply(*rule onorm-le-matrix-component*)
 unfolding *max-norm-def* by(*rule Max-ge*[*OF* *max-norm-set-proptys*]) *force*

0.3.3 Picard Lindeloef for linear systems

Now we prove our first objective. First we obtain the Lipschitz constant for linear systems of ODEs, and then we prove that IVPs arising from these satisfy the conditions for Picard-Lindeloef theorem (hence, they have a unique

solution).

lemma *matrix-lipschitz-constant*:

fixes $A :: \text{real}^{'n} \wedge 'n$

shows $\text{dist } (A * v \ x) \ (A * v \ y) \leq (\text{real } \text{CARD}('n))^2 * (\|A\|_{\text{max}}) * \text{dist } x \ y$

unfolding *dist-norm matrix-vector-mult-diff-distrib[symmetric]*

proof(*subst mult-norm-matrix-sgn-eq[symmetric]*)

have $\|A\|_{\text{op}} \leq (\|A\|_{\text{max}}) * (\text{real } \text{CARD}('n) * \text{real } \text{CARD}('n))$

by (*metis (no-types) Groups.mult-ac(2) op-norm-le-max-norm*)

then have $(\|A\|_{\text{op}}) * (\|x - y\|) \leq (\text{real } \text{CARD}('n))^2 * (\|A\|_{\text{max}}) * (\|x - y\|)$

by (*metis (no-types, lifting) mult.commute mult-right-mono norm-ge-zero power2-eq-square*)

also have $(\|A * v \ \text{sgn } (x - y)\|) * (\|x - y\|) \leq (\|A\|_{\text{op}}) * (\|x - y\|)$

by (*simp add: norm-sgn-le-op-norm mult-mono'*)

ultimately show $(\|A * v \ \text{sgn } (x - y)\|) * (\|x - y\|) \leq (\text{real } \text{CARD}('n))^2 * (\|A\|_{\text{max}}) * (\|x - y\|)$

using *order-trans-rules(23)* **by** *blast*

qed

lemma *picard-lindelof-linear-system*:

fixes $A :: \text{real}^{'n} \wedge 'n$

defines $L \equiv (\text{real } \text{CARD}('n))^2 * (\|A\|_{\text{max}})$

shows *picard-lindelof* $(\lambda t. (*v) \ A) \ \text{UNIV} \ \text{UNIV} \ 0$

apply(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def, clarsimp*)

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

using *max-norm-ge-0[of A]* **unfolding** *assms* **by** *force (rule matrix-lipschitz-constant)*

lemma *picard-lindelof-affine-system*:

fixes $A :: \text{real}^{'n} \wedge 'n$

shows *picard-lindelof* $(\lambda t \ s. A * v \ s + b) \ \text{UNIV} \ \text{UNIV} \ 0$

apply(*rule picard-lindelof-add[OF picard-lindelof-linear-system]*)

using *picard-lindelof-constant* **by** *auto*

0.3.4 Diagonalization

lemma *invertibleI*:

assumes $A ** B = \text{mat } 1$ **and** $B ** A = \text{mat } 1$

shows *invertible* A

using *assms* **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(1)* *invertibleD(2)* *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* :: $('a::\text{semiring-1})^m \times^m \Rightarrow ('a::\text{semiring-1})^n \times^n \Rightarrow \text{bool}$ (*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** $A :: 'a::\text{division-ring}^n \times^n$
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** $A B :: ('a::\text{semiring-1})^n \times n$
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** $A B C :: ('a::\text{semiring-1})^n \times n$
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*: $\text{diag-mat } f * v \ s = (\chi \ i. f \ i * s \$ i)$
unfolding *diag-mat-def matrix-vector-mult-def* **by** *simp*

lemma *matrix-vector-mul-diag-axis*[*simp*]: $\text{diag-mat } f * v \ (\text{axis } i \ k) = \text{axis } i \ (f \ i * k)$
by (*simp add: matrix-vector-mul-diag-mat axis-def fun-eq-iff*)

lemma *matrix-mul-diag-mat*: $\text{diag-mat } f ** A = (\chi \ i \ j. f \ i * A \$ i \$ j)$
unfolding *diag-mat-def matrix-matrix-mult-def* **by** *simp*

lemma *matrix-matrix-mul-diag-mat*: $A ** \text{diag-mat } f = (\chi \ i \ j. A \$ i \$ j * f \ j)$
unfolding *diag-mat-def matrix-matrix-mult-def* **apply**(*clarsimp simp: fun-eq-iff*)

subgoal for $i\ j$ by (auto simp: finite-sum-univ-singleton[of - j])
done

lemma *matrix-mul-diag-diag*: $\text{diag-mat } f ** \text{diag-mat } g = \text{diag-mat } (\lambda i. f\ i * g\ i)$
unfolding *diag-mat-def* *matrix-matrix-mult-def* *vec-eq-iff* by *simp*

lemma *compow-matrix-mul-diag-mat-eq*: $((**) (\text{diag-mat } f) ^n (\text{mat } 1) = \text{diag-mat } (\lambda i. f\ i ^n))$
apply(induct n , simp-all add: *matrix-mul-diag-mat*)
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 $((**) A ^n (\text{mat } 1) = P^{-1} ** (\text{diag-mat } (\lambda i. f\ i ^n)) ** P$
proof(induct n , simp-all add: *assms*)
fix $n::nat$
have $P^{-1} ** \text{diag-mat } f ** P ** (P^{-1} ** \text{diag-mat } (\lambda i. f\ i ^n) ** P) =$
 $P^{-1} ** \text{diag-mat } f ** \text{diag-mat } (\lambda i. f\ i ^n) ** P$ (is ?lhs = -)
by (metis (no-types, lifting) *assms*(1) *invertibleD*(2) *matrix-mul-rid* *matrix-mul-assoc*)

also have $\dots = P^{-1} ** \text{diag-mat } (\lambda i. f\ i * f\ i ^n) ** P$ (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 $((**) A ^n (\text{mat } 1) \sim \text{diag-mat } (\lambda i. f\ i ^n)$
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 $\exists P. \text{invertible } P \wedge ((**) A ^n (\text{mat } 1) = P^{-1} ** \text{diag-mat } (\lambda i. f\ i ^n) ** P$
using *compow-similar-diag-mat-eq* by *blast*
qed

definition *eigen* :: $(\alpha::\text{semiring-1}) ^n \Rightarrow \alpha \Rightarrow \text{bool}$ where
 $\text{eigen } A\ v\ c = (v \neq 0 \wedge A * v = c * v)$

lemma $f\ i \neq 0 \implies \text{eigen } (\text{diag-mat } f) (e\ i) (f\ i)$
unfolding *eigen-def* apply(simp add: *matrix-vector-mul-diag-mat*)
by (simp add: *axis-def* *vector-scalar-mult-def* *fun-eq-iff*)

lemma *sqrt-Max-power2-eq-max-abs*:
 $\text{finite } A \implies A \neq \{\} \implies \text{sqrt } (\text{Max } \{(f\ i)^2 \mid i. i \in A\}) = \text{Max } \{|f\ i| \mid i. i \in A\}$
proof(rule *sym*, subst *cSup-eq-Max[symmetric]*, simp-all, subst *cSup-eq-Max[symmetric]*,
simp-all)

```

assume assms: 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 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 op-norm-diag-mat-eq:  $\| \text{diag-mat } f \|_{op} = \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\}$ 
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 auto
  {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-rsqrt by blast
    hence  $r \leq \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\}$ 
    by (subst (asm) sqrt-Max-power2-eq-max-abs[of UNIV f], simp-all)}
  hence  $1: \forall x \in \{\| \text{diag-mat } f * v\ x \| \mid x. \|x\| = 1\}. x \leq \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\}$ 
  unfolding diag-mat-def by blast
  obtain i where i-def:  $\text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\} = \| \text{diag-mat } f * v\ e\ i \|$ 
  using cMax-finite-ex[of  $\{|f\ i| \mid i. i \in \text{UNIV}\}$ ] by force
  hence  $2: \exists x \in \{\| \text{diag-mat } f * v\ x \| \mid x. \|x\| = 1\}. \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\} \leq x$ 
  by (metis (mono-tags, lifting) abs-1 mem-Collect-eq norm-axis-eq order-refl
real-norm-def)
  show  $\text{Sup } \{\| \text{diag-mat } f * v\ x \| \mid x. \|x\| = 1\} = \text{Max } \{|f\ i| \mid i. i \in \text{UNIV}\}$ 
  by (rule cSup-eq[OF 1 2])
qed

```

```

lemma  $CARD('a) \geq 2 \implies \|diag\text{-}mat\ f\|_{max} = Max\ \{|f\ i|\ |i. i \in UNIV\}$ 
  apply(unfold max-norm-def, simp)
  apply(rule Max-eq-if)
  apply auto
oops

```

```

no-notation matrix-inv ( $-^{-1}$  [90])
  and similar-matrix (infixr  $\sim$  25)

```

0.3.5 Squared matrices

The general solution for linear systems of ODEs involves the an exponential function. Unfortunately, this operation is only available in Isabelle for the type class “banach”. Hence, we define a type of squared matrices and prove that it is an instance of this class.

```

typedef 'm sq-mtx = UNIV::( $real^{m^m}$ ) set
  morphisms to-vec sq-mtx-chi by simp

```

```

declare sq-mtx-chi-inverse [simp]
  and to-vec-inverse [simp]

```

```

setup-lifting type-definition-sq-mtx

```

```

lift-definition sq-mtx-ith :: 'm sq-mtx  $\Rightarrow$  'm  $\Rightarrow$  ( $real^{m^m}$ ) (infixl  $\$$  90) is vec-nth
.

```

```

lift-definition sq-mtx-vec-mult :: 'm sq-mtx  $\Rightarrow$  ( $real^{m^m}$ )  $\Rightarrow$  ( $real^{m^m}$ ) (infixl  $*_V$ 
90)
  is matrix-vector-mult .

```

```

lift-definition vec-sq-mtx-prod :: ( $real^{m^m}$ )  $\Rightarrow$  'm sq-mtx  $\Rightarrow$  ( $real^{m^m}$ ) is vector-matrix-mult
.

```

```

lift-definition sq-mtx-diag :: (('m::finite)  $\Rightarrow$   $real$ )  $\Rightarrow$  ('m::finite) sq-mtx (binder
diag 10) is diag-mat .

```

```

lift-definition sq-mtx-transpose :: ('m::finite) sq-mtx  $\Rightarrow$  'm sq-mtx ( $-^\dagger$ ) is trans-
pose .

```

```

lift-definition sq-mtx-inv :: ('m::finite) sq-mtx  $\Rightarrow$  'm sq-mtx ( $-^{-1}$  [90]) is matrix-inv
.

```

```

lift-definition sq-mtx-row :: 'm  $\Rightarrow$  ('m::finite) sq-mtx  $\Rightarrow$   $real^{m^m}$  (row) is row .

```

```

lift-definition sq-mtx-col :: 'm  $\Rightarrow$  ('m::finite) sq-mtx  $\Rightarrow$   $real^{m^m}$  (col) is column
.

```

```

lemma to-vec-eq-ith: (to-vec A)  $\$$  i = A  $\$$  i

```


by *transfer simp*

lemma *sq-mtx-chi-ith[simp]*: $(sq\text{-mtx-chi } A) \text{ } \$\$ \text{ } i1 \text{ } \$ \text{ } i2 = A \text{ } \$ \text{ } i1 \text{ } \$ \text{ } i2$
by *transfer simp*

lemma *sq-mtx-chi-vec-lambda-ith[simp]*: $sq\text{-mtx-chi } (\chi \text{ } i \text{ } j. x \text{ } i \text{ } j) \text{ } \$\$ \text{ } i1 \text{ } \$ \text{ } i2 = x \text{ } i1 \text{ } i2$
by(*simp add: sq-mtx-ith-def*)

lemma *sq-mtx-eq-iff*:
shows $(\bigwedge i. A \text{ } \$\$ \text{ } i = B \text{ } \$\$ \text{ } i) \implies A = B$
and $(\bigwedge i \text{ } j. A \text{ } \$\$ \text{ } i \text{ } \$ \text{ } j = B \text{ } \$\$ \text{ } i \text{ } \$ \text{ } j) \implies A = B$
by(*transfer, simp add: vec-eq-iff*)**+**

lemma *sq-mtx-diag-simps[simp]*:
 $i = j \implies sq\text{-mtx-diag } f \text{ } \$\$ \text{ } i \text{ } \$ \text{ } j = f \text{ } i$
 $i \neq j \implies sq\text{-mtx-diag } f \text{ } \$\$ \text{ } i \text{ } \$ \text{ } j = 0$
 $sq\text{-mtx-diag } f \text{ } \$\$ \text{ } i = axis \text{ } i \text{ } (f \text{ } i)$
unfolding *sq-mtx-diag-def* **by** (*simp-all add: axis-def vec-eq-iff*)

lemma *sq-mtx-vec-mult-eq*: $m *_{\mathcal{V}} x = (\chi \text{ } i. \text{sum } (\lambda j. (m \text{ } \$\$ \text{ } i \text{ } \$ \text{ } j) * (x \text{ } \$ \text{ } j)))$
UNIV)
by(*transfer, simp add: matrix-vector-mult-def*)

lemma *sq-mtx-transpose-transpose[simp]*: $(A^\dagger)^\dagger = A$
by(*transfer, simp*)

lemma *transpose-mult-vec-canon-row[simp]*: $(A^\dagger) *_{\mathcal{V}} (e \text{ } i) = \text{row } i \text{ } A$
by *transfer (simp add: row-def transpose-def axis-def matrix-vector-mult-def)*

lemma *row-ith[simp]*: $\text{row } i \text{ } A = A \text{ } \$\$ \text{ } i$
by *transfer (simp add: row-def)*

lemma *mtx-vec-mult-canon*: $A *_{\mathcal{V}} (e \text{ } i) = \text{col } i \text{ } A$
by (*transfer, simp add: matrix-vector-mult-basis*)

Squared matrices form a real normed vector space

instantiation *sq-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-ldistrib 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-ith*[simp]: $(A + B) \ \$\$ i = A \ \$\$ i + B \ \$\$ i$
by(*unfold plus-sq-mtx-def*, *transfer*, *simp*)

lemma *sq-mtx-minus-ith*[simp]: $(A - B) \ \$\$ i = A \ \$\$ i - B \ \$\$ i$
by(*unfold minus-sq-mtx-def*, *transfer*, *simp*)

lemma *sq-mtx-plus-diag-diag*[simp]: $\text{sq-mtx-diag } f + \text{sq-mtx-diag } g = (\text{diag } i. f \ i + g \ i)$
by (*rule sq-mtx-eq-iff*(2)) (*simp add: axis-def*)

lemma *sq-mtx-minus-diag-diag*[simp]: $\text{sq-mtx-diag } f - \text{sq-mtx-diag } g = (\text{diag } i. f \ i - g \ i)$
by (*rule sq-mtx-eq-iff*(2)) (*simp add: axis-def*)

lemma *sum-sq-mtx-diag*[simp]: $(\sum n < m. \text{sq-mtx-diag } (g \ n)) = (\text{diag } i. \sum n < m. (g \ n \ i))$ **for** $m :: \text{nat}$
by (*induct m*, *simp*, *rule sq-mtx-eq-iff*, *simp-all*)

lemma *sq-mtx-mult-diag-diag*[simp]: $\text{sq-mtx-diag } f * \text{sq-mtx-diag } g = (\text{diag } i. f \ i * g \ i)$
by (*simp add: matrix-mul-diag-diag sq-mtx-diag.abs-eq times-sq-mtx.abs-eq*)

lemma *sq-mtx-diag-vec-mult*: $\text{sq-mtx-diag } f *_{\text{V}} s = (\chi \ i. f \ i * s \ \$ i)$
by (*simp add: matrix-vector-mul-diag-mat sq-mtx-diag.abs-eq sq-mtx-vec-mult.abs-eq*)

lemma *sq-mtx-mult-diagl*: $\text{sq-mtx-diag } f * A = \text{sq-mtx-chi } (\chi \ i \ j. f \ i * A \ \$\$ i \ \$ j)$
by *transfer (simp add: matrix-mul-diag-mat)*

lemma *sq-mtx-mult-diagr*: $A * \text{sq-mtx-diag } f = \text{sq-mtx-chi } (\chi \ i \ j. A \ \$\$ i \ \$ j * f \ j)$
by *transfer (simp add: matrix-matrix-mul-diag-mat)*

lemma *mtx-vec-mult-0l*[simp]: $0 *_{\text{V}} x = 0$

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*)

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*: $\{0 < ..\}$. *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-zero-iff)
by(transfer, simp add: norm-sq-mtx-def op-norm-scaleR algebra-simps)+

end

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 sq-mtx-diag f = (diag i. c * f i)
by (rule sq-mtx-eq-iff(2), 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 mtrx-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 le-mtx-norm: m ∈ {||A *V x|| | x. ||x|| = 1} ⇒ m ≤ ||A||
using cSup-upper[of - {||(to-vec A) *V x|| | 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|| ≤ (||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 (λA s. A *V s)
apply (rule bounded-bilinear.intro, simp-all add: mtrx-vec-mult-add-rdistr
  mtrx-vec-mult-add-rdistl scaleR-mtx-vec-assoc mtrx-vec-scaleR-commute)
by (rule-tac x=1 in exI, auto intro!: norm-vec-mult-le)

lemma norm-sq-mtx-def2: ||A|| = Sup {||A *V x|| | x. ||x|| = 1}
unfolding norm-sq-mtx-def op-norm-def sq-mtx-vec-mult-def by simp

lemma norm-sq-mtx-def3: ||A|| = SUPREMUM UNIV (λx. (||A *V x||) / (||x||))
unfolding norm-sq-mtx-def onorm-def sq-mtx-vec-mult-def by simp

lemma norm-sq-mtx-diag: ||sq-mtx-diag f|| = Max {|f i| | i. i ∈ UNIV}
unfolding norm-sq-mtx-def apply transfer
by (rule op-norm-diag-mat-eq)

lemma sq-mtx-norm-le-sum-col: ||A|| ≤ (∑ i ∈ UNIV. ||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|| ≤ ||A†||
unfolding norm-sq-mtx-def by transfer (rule op-norm-le-transpose)

lemma norm-eq-norm-transpose[simp]: ||A†|| = ||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 e i] **by** *simp*

Squared matrices form a Banach space

instantiation *sq-mtx* :: (*finite*) *real-normed-algebra-1*
begin

lift-definition *one-sq-mtx* :: 'a *sq-mtx* **is** *sq-mtx-chi* (*mat* 1) .

lemma *sq-mtx-one-idty*: $1 * A = A \ A * 1 = A$ **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: sq-mtx-chi-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*, *rule sq-mtx-eq-iff*(2), *simp add: axis-def*)+

lemma *mtx-vec-mult-1*[*simp*]: $1 *_{\mathcal{V}} s = s$
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 (*rule sq-mtx-eq-iff*(2), *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*: *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*, *clar-simp*, *safe*)

apply(*rule-tac* $x = \text{sq-mtx-chi } A' \text{ in } exI$, *simp*)
by (*rule-tac* $x = \text{to-vec } A' \text{ in } exI$, *simp* *add: sq-mtx-chi-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*)
by (*smt* *UNIV-I* *sq-mtx-chi-inverse* *sq-mtx-inv.abs-eq* *times-sq-mtx.abs-eq* *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-symm*: $A \sim B \implies B \sim A$
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*: $(sq\text{-}mtx\text{-}diag\ f)^{\wedge} n = (diag\ i.\ f\ i^{\wedge} n)$
by (*induct n*, *simp-all*)

lemma *power-similar-sq-mtx-diag-eq*:
assumes *mtx-invertible P*
and $A = P^{-1} * (sq\text{-}mtx\text{-}diag\ f) * P$
shows $A^{\wedge} n = P^{-1} * (diag\ i.\ f\ i^{\wedge} n) * P$
proof(*induct n*, *simp-all add: assms*)
fix $n::nat$
have $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (diag\ i.\ f\ i^{\wedge} n) * P) =$
 $P^{-1} * sq\text{-}mtx\text{-}diag\ f * (diag\ i.\ f\ i^{\wedge} n) * P$
by (*metis (no-types, lifting) assms(1) sign-simps(4) mtx-invertibleD(2) sq-mtx-one-idty(2)*)
also have $\dots = P^{-1} * (diag\ i.\ f\ i * f\ i^{\wedge} n) * P$
by (*simp add: mult.assoc*)
finally show $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (diag\ i.\ f\ i^{\wedge} n) * P) =$
 $P^{-1} * (diag\ i.\ f\ i * f\ i^{\wedge} n) * P .$

qed

lemma *power-similar-sq-mtx-diag*:
assumes $A \sim (sq\text{-}mtx\text{-}diag\ f)$
shows $A^{\wedge} n \sim (diag\ i.\ f\ i^{\wedge} n)$
using *assms power-similar-sq-mtx-diag-eq*
unfolding *similar-sq-mtx-def* **by** *blast*

lemma *Cauchy-cols*:
fixes $X :: nat \Rightarrow ('a::finite)\ sq\text{-}mtx$
assumes *Cauchy X*
shows *Cauchy* $(\lambda n.\ col\ i\ (X\ n))$
proof(*unfold Cauchy-def dist-norm, clarsimp*)
fix $\varepsilon::real$ **assume** $\varepsilon > 0$
then obtain M **where** $M\text{-def}:\forall m \geq M. \forall n \geq M. \|X\ m - X\ n\| < \varepsilon$
using $\langle Cauchy\ X \rangle$ **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 $X \longrightarrow \text{sq-mtx-chi } (\text{transpose } L)$
proof(*unfold LIMSEQ-def dist-norm, clarsimp*)
 let $?L = \text{sq-mtx-chi } (\text{transpose } L)$
 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-def* **by** *blast*
 hence $(\sum_{i \in \text{UNIV}. \|\text{col } i (X n - ?L)\|}) < (\sum_{(i :: 'a) \in \text{UNIV}. \varepsilon / ?a)$
 using *sum-strict-mono*[*of* $\lambda i. \|\text{col } i (X n - ?L)\|$] **by** *force*
 moreover have $\|X n - ?L\| \leq (\sum_{i \in \text{UNIV}. \|\text{col } i (X n - ?L)\|)$
 using *sq-mtx-norm-le-sum-col* **by** *blast*
 moreover have $(\sum_{(i :: 'a) \in \text{UNIV}. \varepsilon / ?a) = \varepsilon$
 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 $X :: \text{nat} \Rightarrow 'a \text{ sq-mtx}$
 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 $A = P^{-1} * (\text{sq-mtx-diag } f) * P$
shows $\text{exp } A = P^{-1} * \text{exp } (\text{sq-mtx-diag } f) * P$
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 $\text{exp } A \sim \text{exp } (\text{sq-mtx-diag } f)$
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 $r :: \text{real}$ **assume** $r > 0$
have $\forall i. \exists n_0. \forall n \geq n_0. ?g n i < r$
using *assms* $\langle r > 0 \rangle$ **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 $n :: \text{nat}$
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* $\langle n \geq N \rangle$ **by** *blast*
hence $\text{Max } \{x. \exists i. x = ?g n i\} < r$

unfolding $i\text{-def}[\textit{symmetric}] \cdot \}$
thus $\exists N. \forall n \geq N. \text{Max} \{x. \exists i. x = ?g\ n\ i\} < r$
by *blast*
qed

lemma *exp-sq-mtx-diag*: $\text{exp}(\text{sq-mtx-diag } f) = (\text{diag } i. \text{exp}(f\ i))$
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 *has-derivative-mtx-ith*[*derivative-intros*]:
fixes $t :: \textit{real}$ **and** $T :: \textit{real set}$
defines $t_0 \equiv \textit{netlimit}(\text{at } t \text{ within } T)$
assumes $D\ A \mapsto (\lambda h. h *_{\text{R}} A' t)$ *at* t *within* T
shows $D(\lambda t. A\ t\ \$\$ i) \mapsto (\lambda h. h *_{\text{R}} A' t\ \$\$ i)$ *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= $\lambda \tau. (A\ \tau - A\ t_0 - (\tau - t_0) *_{\text{R}} A' t) /_{\text{R}} (\|\tau - t_0\|)$ 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*[*simp, derivative-intros*] =
bounded-bilinear.FDERIV[OF bounded-bilinear-sq-mtx-vec-mult]

lemma *vderiv-mtx-vec-mult-intro*[*poly-derivatives*]:
assumes $D\ u = u'$ *on* T **and** $D\ A = A'$ *on* T
and $g = (\lambda t. A\ t *_{\text{V}} u' t + A' t *_{\text{V}} u\ t)$
shows $D(\lambda t. A\ t *_{\text{V}} u\ t) = g$ *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(146)*)
by (*auto simp: fun-eq-iff mtrx-vec-scaleR-commute pth-6 scaleR-mtx-vec-assoc*)

lemma *has-derivative-mtx-vec-multl*[*derivative-intros*]:
assumes $\bigwedge i\ j. D(\lambda t. (A\ t)\ \$\$ i\ \$ j) \mapsto (\lambda \tau. \tau *_{\text{R}} (A' t)\ \$\$ i\ \$ j)$ *(at* t *within* $T)$
shows $D(\lambda t. A\ t *_{\text{V}} x) \mapsto (\lambda \tau. \tau *_{\text{R}} (A' t) *_{\text{V}} x)$ *at* t *within* T
unfolding *sq-mtx-vec-mult-sum-cols*
apply(*rule-tac f'1= $\lambda i\ \tau. \tau *_{\text{R}} (x\ \$ i *_{\text{R}} \text{col } i\ (A' t))$ in derivative-eq-intros(9)*)
apply(*simp-all add: scaleR-right.sum*)
apply(*rule-tac g'1= $\lambda \tau. \tau *_{\text{R}} \text{col } i\ (A' t)$ 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*: $D A = A'$ on $T \implies$ *continuous-on* T ($\lambda \tau. A \tau *_{\mathcal{V}} b$)

apply(rule *vderiv-on-continuous-on*[*OF vderiv-mtx-vec-mult-intro*])
by (rule *derivative-intros*, *auto*)

lemma *continuous-on-mtx-vec-multl*: *continuous-on* S $((*_V) A)$
by *transfer* (*simp add: matrix-vector-mult-linear-continuous-on*)

— Automatically generated derivative rules from this subsubsection

thm *derivative-eq-intros*(145,146,147)

0.3.6 Flow for squared matrix systems

Finally, we can use the *exp* operation to characterize the general solutions for linear systems of ODEs. We show that they satisfy the *local-flow* locale.

lemma *picard-lindelof-sq-mtx-linear*:

fixes $A :: ('n::\text{finite}) \text{ sq-mtx}$
defines $L \equiv (\text{real } \text{CARD}('n))^2 * (\| \text{to-vec } A \|_{\max})$
shows *picard-lindelof* $(\lambda t s. A *_V s)$ *UNIV UNIV* t_0
apply(*unfold-locales*, *simp-all add: local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(rule-tac $x=1$ **in** *exI*, *clarsimp*, rule-tac $x=L$ **in** *exI*, *safe*)
using *max-norm-ge-0*[*of to-vec A*] **unfolding** *assms* **apply** *force*
by *transfer* (rule *matrix-lipschitz-constant*)

lemma *picard-lindelof-sq-mtx-affine*:

fixes $A :: ('n::\text{finite}) \text{ sq-mtx}$
shows *picard-lindelof* $(\lambda t s. A *_V s + b)$ *UNIV UNIV* t_0
apply(rule *picard-lindelof-add*[*OF picard-lindelof-sq-mtx-linear*])
using *picard-lindelof-constant* **by** *auto*

lemma *local-flow-sq-mtx-linear*:

fixes $A :: ('n::\text{finite}) \text{ sq-mtx}$
shows *local-flow* $((*_V) A)$ *UNIV UNIV* $(\lambda t s. \text{exp } (t *_R A) *_V s)$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindelof-sq-mtx-linear* **apply** *blast*
apply(rule *vderiv-mtx-vec-mult-intro*, rule *poly-derivatives*)
by (rule *has-vderiv-on-exp-scaleRl*) (*auto simp: fun-eq-iff*
exp-times-scaleR-commute sq-mtx-times-vec-assoc intro: poly-derivatives)

lemma *local-flow-sq-mtx-affine*:

fixes $A :: ('n::\text{finite}) \text{ sq-mtx}$
shows *local-flow* $(\lambda s. A *_V s + b)$ *UNIV UNIV*
 $(\lambda t s. (\text{exp } (t *_R A)) *_V s + (\text{exp } (t *_R A)) *_V \text{ivl-integral } 0 t (\lambda \tau. (\text{exp } (- \tau *_R A)) *_V b))$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindelof-sq-mtx-affine* **apply** *blast*
apply(*intro poly-derivatives*, rule *poly-derivatives*, rule *poly-derivatives*, *force*, *force*)

```

apply(rule ivl-integral-has-vderiv-on[OF continuous-on-mtx-vec-multl])
apply(intro poly-derivatives, simp-all add: mtx-vec-mult-add-rdistl)
unfolding sq-mtx-times-vec-assoc[symmetric]
by (auto intro: poly-derivatives simp: exp-minus-inverse exp-times-scaleR-commute)

lemma local-flow-sq-mtx-diag-linear:
  fixes A :: ('n::finite) sq-mtx
  assumes mtx-invertible P
  and A = P-1 * (sq-mtx-diag f) * P
  shows local-flow ((*V) A) UNIV UNIV (λt s. (P-1 * (diag i. exp (t * f i)) *
P) *V s)
proof -
  {fix t 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 have exp (t *R A) = P-1 * (diag i. exp (t * f i)) * P .}
  hence ∧ t s. exp (t *R A) *V s = (P-1 * (diag i. exp (t * f i)) * P) *V s
    by force
  thus ?thesis
    using local-flow-sq-mtx-linear[of A] by force
qed

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-prelims-dyn-sys

```

```

begin

```

```

type-synonym 'a pred = 'a ⇒ bool

```

```

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

```

```

notation Union (μ)
and g-orbital ((1x' =- & - on - - @ -))

```

```

abbreviation skip ≡ (λs. {s})

```

0.4.1 Verification of regular programs

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

definition $fbox :: ('a \Rightarrow 'b \text{ set}) \Rightarrow 'b \text{ pred} \Rightarrow 'a \text{ pred} \text{ (|·| - [61,81] 82)}$
where $|F| P = (\lambda s. (\forall s'. s' \in F s \longrightarrow P s'))$

lemma $fbox\text{-}iso: P \leq Q \Longrightarrow |F| P \leq |F| Q$
unfolding $fbox\text{-}def$ **by** $auto$

lemma $fbox\text{-}invariants$:
assumes $I \leq |F| I$ **and** $J \leq |F| J$
shows $(\lambda s. I s \wedge J s) \leq |F| (\lambda s. I s \wedge J s)$
and $(\lambda s. I s \vee J s) \leq |F| (\lambda s. I s \vee J s)$
using $assms$ **unfolding** $fbox\text{-}def$ **by** $auto$

Now, we compute wlp for specific programs.

lemma $fbox\text{-}eta[simp]: fbox \text{ skip } P = P$
unfolding $fbox\text{-}def$ **by** $simp$

Next, we introduce assignments and their wlp.

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

definition $assign :: 'n \Rightarrow ('a \wedge 'n \Rightarrow 'a) \Rightarrow 'a \wedge 'n \Rightarrow ('a \wedge 'n) \text{ set} \text{ ((2- ::= -) [70, 65] 61)}$
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** $(subst fbox\text{-}def) simp$

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

definition $kcomp :: ('a \Rightarrow 'b \text{ set}) \Rightarrow ('b \Rightarrow 'c \text{ set}) \Rightarrow ('a \Rightarrow 'c \text{ set}) \text{ (infixl ; 75)}$
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 $fbox\text{-}kcomp\text{-}ge$:
assumes $P \leq |G| R$ $R \leq |F| Q$
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 $ifthenelse :: 'a \text{ pred} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow ('a \Rightarrow 'b \text{ set})$
 $(IF - THEN - ELSE - [64,64,64] 63)$ **where**
 $IF P THEN X ELSE Y \equiv (\lambda s. \text{if } P s \text{ then } X s \text{ else } Y s)$

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

lemma *hoare-if-then-else*:
assumes $(\lambda s. P\ s \wedge T\ s) \leq |X|\ Q$
and $(\lambda s. P\ s \wedge \neg\ T\ s) \leq |Y|\ Q$
shows $P \leq |IF\ T\ THEN\ X\ ELSE\ Y|\ Q$
using *assms* **unfolding** *fbx-def ifthenelse-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-base*:
shows $kpower\ f\ 0\ s = \{s\}$ **and** $kpower\ f\ (Suc\ 0)\ s = f\ s$
unfolding *kpower-def* **by**(*auto simp: kcomp-eq*)

lemma *kpower-simp*: $kpower\ f\ (Suc\ n)\ s = (f\ ;\ kpower\ f\ n)\ s$
unfolding *kcomp-eq* **apply**(*induct n*)
unfolding *kpower-base* **apply**(*rule subset-antisym, clarsimp, force, clarsimp*)
unfolding *kpower-def kcomp-eq* **by** *simp*

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

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

lemma *kstar-inv*: $I \leq |F|\ I \Longrightarrow I \leq |F^*|\ I$
unfolding *kleene-star-def fbx-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 *fbx-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*: $P \leq I \Rightarrow I \leq Q \Rightarrow I \leq |F| I \Rightarrow P \leq |LOOP F INV I| Q$
unfolding *loopi-def* **using** *fbox-kstarI*[*of P*] **by** *simp*

0.4.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

context *local-flow*
begin

lemma *fbox-g-ode*: $|x' = f \ \& \ G \text{ on } T S @ 0| Q =$
 $(\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$ (**is** $= ?wlp$)
unfolding *fbox-g-orbital* **apply**(*rule ext, safe, clarsimp*)
apply(*erule-tac x = \lambda t. \varphi t s in ballE*)
using *in-ivp-sols* **apply**(*force, force, force simp: init-time ivp-sols-def*)
apply(*subgoal-tac \forall \tau \in \text{down } T t. X \tau = \varphi \tau s, simp-all, clarsimp*)
apply(*subst eq-solution, simp-all add: ivp-sols-def*)
using *init-time* **by** *auto*

lemma *fbox-g-ode-ivl*: $t \geq 0 \Rightarrow t \in T \Rightarrow |x' = f \ \& \ G \text{ on } \{0..t\} S @ 0| Q =$
 $(\lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)))$
unfolding *fbox-g-orbital* **apply**(*rule ext, clarsimp, safe*)
apply(*erule-tac x = \lambda t. \varphi t s in ballE, force*)
using *in-ivp-sols-ivl* **apply**(*force simp: closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp: ivp-sols-def*)

```

apply(subgoal-tac  $\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. X \tau = \varphi \tau s)$ , simp, clarsimp)
apply(subst eq-solution-ivl, simp-all add: ivp-sols-def)
  apply(rule has-vderiv-on-subset, force, force simp: closed-segment-eq-real-ivl)
  apply(force simp: closed-segment-eq-real-ivl)
using interval-time init-time apply (meson is-interval-1 order-trans)
using init-time by force

```

```

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 :: (('a::banach)  $\Rightarrow$  'a)  $\Rightarrow$  'a pred  $\Rightarrow$  real set  $\Rightarrow$  'a set  $\Rightarrow$ 
  real  $\Rightarrow$  'a pred  $\Rightarrow$  ('a  $\Rightarrow$  'a set) (( $1x' = -$  &  $-$  on  $- - @ -$  DINV  $-$  ))
  where ( $x' = f$  &  $G$  on  $T S @ t_0$  DINV  $I$ ) = ( $x' = f$  &  $G$  on  $T S @ t_0$ )

```

```

lemma fbox-g-orbital-guard:
  assumes  $H = (\lambda s. G s \wedge Q s)$ 
  shows  $|x' = f$  &  $G$  on  $T S @ t_0] \ Q = |x' = f$  &  $G$  on  $T S @ t_0] \ H$ 
  unfolding fbox-g-orbital using assms by auto

```

```

lemma fbox-g-orbital-inv:
  assumes  $P \leq I$  and  $I \leq |x' = f$  &  $G$  on  $T S @ t_0] \ I$  and  $I \leq Q$ 
  shows  $P \leq |x' = f$  &  $G$  on  $T S @ t_0] \ Q$ 
  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]:
  ( $I \leq |x' = f$  &  $G$  on  $T S @ t_0] \ I$ ) = diff-invariant  $I$  f  $T S t_0 \ G$ 
  by (auto simp: diff-invariant-def ivp-sols-def fbox-def g-orbital-eq)

```

```

lemma diff-inv-guard-ignore:
  assumes  $I \leq |x' = f$  &  $(\lambda s. \text{True})$  on  $T S @ t_0] \ I$ 
  shows  $I \leq |x' = f$  &  $G$  on  $T S @ t_0] \ I$ 
  using assms unfolding fbox-diff-inv diff-invariant-eq by auto

```

context local-flow

begin

```

lemma fbox-diff-inv-eq: diff-invariant  $I$  f  $T S 0 \ (\lambda s. \text{True}) =$ 
  ( $(\lambda s. s \in S \longrightarrow I s) = |x' = f$  &  $(\lambda s. \text{True})$  on  $T S @ 0] \ (\lambda s. s \in S \longrightarrow I s)$ )
  unfolding fbox-diff-inv[symmetric] fbox-g-orbital le-fun-def fun-eq-iff
  using init-time apply(clarsimp simp: subset-eq ivp-sols-def)
  apply(safe, force, force)
  apply(subst ivp(2)[symmetric], simp)
  apply(erule-tac  $x = \lambda t. \varphi t x$  in allE)
  using in-domain has-vderiv-on-domain ivp(2) init-time by auto

```


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

end

lemma *fbox-g-odei*: $P \leq I \implies I \leq |x' = f \ \& \ G \text{ on } T S @ t_0| I \implies (\lambda s. I s \wedge G s) \leq Q \implies$
 $P \leq |x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I| Q$
unfolding *g-ode-inv-def* **apply**(*rule-tac* $b = |x' = f \ \& \ G \text{ on } T S @ t_0| I$ **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.

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

lemma *diff-solve-rule*:
assumes *local-flow* $f T \text{ UNIV } \varphi$
and $\forall s. P s \longrightarrow (\forall t \in T. (\mathcal{P} (\lambda t. \varphi t s) (\text{down } T t) \subseteq \{s. G s\}) \longrightarrow Q (\varphi t s))$
shows $P \leq |x' = f \ \& \ G \text{ on } T \text{ UNIV } @ 0| Q$
using *assms* **by**(*subst local-flow.fbox-g-ode*) *auto*

lemma *diff-weak-axiom*: $|x' = f \ \& \ G \text{ on } T S @ t_0| Q = |x' = f \ \& \ G \text{ on } T S @ t_0| (\lambda s. G s \longrightarrow Q s)$
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 $|x' = f \ \& \ G \text{ on } T S @ t_0| C = (\lambda s. True)$
and $\forall \tau \in (\text{down } T t). x \tau \in (x' = f \ \& \ G \text{ on } T S @ t_0) s$
shows $\forall \tau \in (\text{down } T t). C (x \tau)$
proof
fix τ **assume** $\tau \in (\text{down } T t)$

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

lemma *diff-cut-axiom*:

assumes *Thyp*: *is-interval* $T \ t_0 \in T$
 and $|x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0| \ C = (\lambda s. \text{True})$
 shows $|x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0| \ Q = |x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0|$
 Q
 proof(*rule-tac* $f = \lambda x. |x| \ Q$ in *HOL.arg-cong*, *rule ext*, *rule subset-antisym*)
 fix s
 {fix s' assume $s' \in (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ s$
 then obtain $\tau :: \text{real}$ and X where $x\text{-ivp}$: $X \in \text{Sols } (\lambda t. f) \ T \ S \ t_0 \ s$
 and $X \ \tau = s'$ and $\tau \in T$ and $\text{guard-}x$: $\mathcal{P} \ X \ (\text{down } T \ \tau) \subseteq \{s. G \ s\}$
 using *g-orbitalD*[*of* $s' \ f \ G \ T \ S \ t_0 \ s$] by *blast*
 have $\forall t \in (\text{down } T \ \tau). \ \mathcal{P} \ X \ (\text{down } T \ t) \subseteq \{s. G \ s\}$
 using *guard-x* by (*force simp*: *image-def*)
 also have $\forall t \in (\text{down } T \ \tau). \ t \in T$
 using $\langle \tau \in T \rangle$ *Thyp* *closed-segment-subset-interval* by *auto*
 ultimately have $\forall t \in (\text{down } T \ \tau). \ X \ t \in (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ s$
 using *g-orbitalI*[*OF* $x\text{-ivp}$] by (*metis* (*mono-tags*, *lifting*))
 hence $\forall t \in (\text{down } T \ \tau). \ C \ (X \ t)$
 using *assms(3)* unfolding *fbox-def* by *meson*
 hence $s' \in (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0) \ s$
 using *g-orbitalI*[*OF* $x\text{-ivp}$ $\langle \tau \in T \rangle$] *guard-x* $\langle X \ \tau = s' \rangle$ by *fastforce*}
 thus $(x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0) \ s$
 by *blast*
 next show $\bigwedge s. (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0) \ s \subseteq (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ s$
 by (*auto simp*: *g-orbital-eq*)
 qed

lemma *diff-cut-rule*:

assumes *Thyp*: *is-interval* $T \ t_0 \in T$
 and *fbox-C*: $P \leq |x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0| \ C$
 and *fbox-Q*: $P \leq |x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0| \ Q$
 shows $P \leq |x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0| \ Q$
 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 $t \in T$
 and $x\text{-ivp}$: $X \in \text{Sols } (\lambda t. f) \ T \ S \ t_0 \ s$
 and *guard-x*: $\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G \ (X \ \tau)$
 have $\forall \tau \in (\text{down } T \ t). \ X \ \tau \in (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ s$
 using *g-orbitalI*[*OF* $x\text{-ivp}$] *guard-x* by *auto*
 hence $\forall \tau \in (\text{down } T \ t). \ C \ (X \ \tau)$
 using *fbox-C* $\langle P \ s \rangle$ by (*subst* (*asm*) *fbox-def*, *auto*)
 hence $X \ t \in (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0) \ s$

```

using guard-x  $\langle t \in T \rangle$  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

```

The rules of dL

abbreviation *g-global-orbit* :: $((a::\text{banach}) \Rightarrow 'a) \Rightarrow 'a\ \text{pred} \Rightarrow 'a \Rightarrow 'a\ \text{set}$
 $((1x' = - \ \& \ -))$ **where** $(x' = f \ \& \ G) \equiv (x' = f \ \& \ G\ \text{on}\ UNIV\ UNIV\ @\ 0)$

abbreviation *g-global-ode-inv* :: $((a::\text{banach}) \Rightarrow 'a) \Rightarrow 'a\ \text{pred} \Rightarrow 'a\ \text{pred} \Rightarrow 'a \Rightarrow 'a\ \text{set}$
 $((1x' = - \ \& \ -\ DINV\ -))$ **where** $(x' = f \ \& \ G\ DINV\ I) \equiv (x' = f \ \& \ G\ \text{on}\ UNIV\ UNIV\ @\ 0\ DINV\ I)$

lemma *solve*:

```

assumes local-flow f UNIV UNIV  $\varphi$ 
and  $\forall s. P\ s \longrightarrow (\forall t. (\forall \tau \leq t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$ 
shows  $P \leq |x' = f \ \& \ G| Q$ 
apply(rule diff-solve-rule[OF assms(1)])
using assms(2) by simp

```

lemma *DS*:

```

fixes c::'a::\{heine-borel, banach\}
shows  $|x' = (\lambda s. c) \ \& \ G| Q = (\lambda x. \forall t. (\forall \tau \leq t. G\ (x + \tau *_R c)) \longrightarrow Q\ (x + t *_R c))$ 
by (subst diff-solve-axiom[of UNIV]) auto

```

lemma *DW*: $|x' = f \ \& \ G| Q = |x' = f \ \& \ G| (\lambda s. G\ s \longrightarrow Q\ s)$
by (*rule diff-weak-axiom*)

lemma *dW*: $G \leq Q \Longrightarrow P \leq |x' = f \ \& \ G| Q$
by (*rule diff-weak-rule*)

lemma *DC*:

```

assumes  $|x' = f \ \& \ G| C = (\lambda s. True)$ 
shows  $|x' = f \ \& \ G| Q = |x' = f \ \& \ (\lambda s. G\ s \wedge C\ s)| Q$ 
by (rule diff-cut-axiom) (auto simp: assms)

```

lemma *dC*:

```

assumes  $P \leq |x' = f \ \& \ G| C$ 
and  $P \leq |x' = f \ \& \ (\lambda s. G\ s \wedge C\ s)| Q$ 
shows  $P \leq |x' = f \ \& \ G| Q$ 
apply(rule diff-cut-rule)
using assms by auto

```

lemma *dI*:

```

assumes  $P \leq I$  and diff-invariant I f UNIV UNIV 0 G and  $I \leq Q$ 
shows  $P \leq |x' = f \ \& \ G| Q$ 
by (rule fbox-g-orbital-inv[OF assms(1) - assms(3)]) (simp add: 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-prelims-matrices hs-vc-spartan
```

begin

Preliminary preparation for the examples.

— Finite set of program variables.

```
typedef program-vars = {"x", "y"}
morphisms to-str to-var
apply(rule-tac x="x" in exI)
by simp
```

```
notation to-var ( $\downarrow_V$ )
```

```
lemma number-of-program-vars:  $CARD(program-vars) = 2$ 
using type-definition.card type-definition-program-vars by fastforce
```

```
instance program-vars::finite
apply(standard, subst bij-betw-finite[of to-str UNIV {"x", "y"}])
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 program-vars-univ-eq:  $(UNIV::program-vars\ set) = \{\downarrow_V "x", \downarrow_V "y"\}$ 
apply auto by (metis to-str to-str-inverse insertE singletonD)
```

```
lemma program-vars-exhaust:  $x = \downarrow_V "x" \vee x = \downarrow_V "y"$ 
using program-vars-univ-eq by auto
```

```
abbreviation val-p ::  $real^{program-vars} \Rightarrow string \Rightarrow real$  (infixl  $\downarrow_V$  90)
where  $store \downarrow_V var \equiv store \$ \downarrow_V var$ 
```

Circular Motion

— Verified with differential invariants.

```
abbreviation circular-motion-vec-field ::  $real^{program-vars} \Rightarrow real^{program-vars}$ 
(C)
where circular-motion-vec-field  $s \equiv (\chi\ i.\ if\ i = \downarrow_V "x" then\ s \downarrow_V "y" else\ -s \downarrow_V "x")$ 
```

lemma *circular-motion-invariants*:

$(\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2) \leq |x'=C \ \& \ G] \ (\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2)$
by (*auto intro!*: *diff-invariant-rules poly-derivatives simp: to-var-inject*)

— Verified with the flow.

abbreviation *circular-motion-flow* :: $real \Rightarrow real^{\wedge}program\text{-}vars \Rightarrow real^{\wedge}program\text{-}vars$
 (φ_C)

where $\varphi_C \ t \ s \equiv (\chi \ i. \text{if } i = |_V''x'' \text{ then } s|_V''x'' * \cos t + s|_V''y'' * \sin t$
 $\text{else } -s|_V''x'' * \sin t + s|_V''y'' * \cos t)$

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV φ_C*

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 program-vars-univ-eq to-var-inject power2-commute*)
apply(*clarsimp, case-tac i = |_V''x''*)
using *program-vars-exhaust by (force intro!: poly-derivatives simp: to-var-inject)+*

lemma *circular-motion*:

$(\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2) \leq |x'=C \ \& \ G] \ (\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2)$
by (*force simp: local-flow.fbox-g-ode[OF local-flow-circ-motion] to-var-inject*)

— Verified by providing dynamics.

lemma *circular-motion-dyn*:

$(\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2) \leq |EVOL \ \varphi_C \ G \ T] \ (\lambda s. r^2 = (s|_V''x'')^2 + (s|_V''y'')^2)$
by (*force simp: to-var-inject*)

no-notation *circular-motion-vec-field* (*C*)
and *circular-motion-flow* (φ_C)

— Verified as a linear system (using uniqueness).

abbreviation *circular-motion-sq-mtx* :: $2 \text{ sq-mtx } (C)$

where $C \equiv \text{sq-mtx-chi } (\chi \ i. \text{if } i = 1 \text{ then } -e \ 2 \text{ else } e \ 1)$

abbreviation *circular-motion-mtx-flow* :: $real \Rightarrow real^{\wedge}2 \Rightarrow real^{\wedge}2 \ (\varphi_C)$

where $\varphi_C \ t \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } s\$1 * \cos t - s\$2 * \sin t \text{ else } s\$1 * \sin t + s\$2 * \cos t)$

lemma *circular-motion-mtx-exp-eq*: $\exp(t *_R C) *_V s = \varphi_C \ t \ s$

apply(*rule local-flow.eq-solution[OF local-flow-sq-mtx-linear, symmetric]*)
apply(*rule ivp-solsI, simp add: sq-mtx-vec-mult-def matrix-vector-mult-def*)

apply(*force intro!*: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *circular-motion-sq-mtx*:

$(\lambda s. r^2 = (s\$1)^2 + (s\$2)^2) \leq \text{fbox } (x' = (*_V) C \ \& \ G) (\lambda s. r^2 = (s\$1)^2 + (s\$2)^2)$

unfolding *local-flow.fbox-g-ode*[*OF local-flow-sq-mtx-linear*] *circular-motion-mtx-exp-eq*
by *auto*

no-notation *circular-motion-sq-mtx* (*C*)
and *circular-motion-mtx-flow* (φ_C)

Bouncing Ball

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:

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

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

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

abbreviation *cnst-acc-vec-field* :: *real* \Rightarrow *real* $\hat{\text{program-vars}}$ \Rightarrow *real* $\hat{\text{program-vars}}$ (*K*)

where $K \ a \ s \equiv (\chi \ i. \text{if } i = (\downarrow_V''x'') \text{ then } s \downarrow_V''y'' \text{ else } a)$

lemma *bouncing-ball-invariants*:

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

$(\lambda s. s \downarrow_V''x'' = h \wedge s \downarrow_V''y'' = 0) \leq \text{fbox}$

(*LOOP*

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

$(\text{IF } (\lambda s. s \downarrow_V''x'' = 0) \text{ THEN } (\downarrow_V''y'' ::= (\lambda s. - s \downarrow_V''y'')) \text{ ELSE skip}))$

$\text{INV } (\lambda s. s \downarrow_V''x'' \geq 0 \wedge 2 * g * s \downarrow_V''x'' - 2 * g * h - (s \downarrow_V''y'' * s \downarrow_V''y'') = 0))$

$(\lambda s. 0 \leq s \downarrow_V''x'' \wedge s \downarrow_V''x'' \leq h)$

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

apply(*force*, *force simp: bb-real-arith*)
by (*rule fbox-g-odei*) (*auto intro!: poly-derivatives diff-invariant-rules simp: to-var-inject*)

— Verified with the flow.

lemma *picard-lindeloeuf-cnst-acc*:
fixes *g::real*
shows *picard-lindeloeuf* ($\lambda t. K\ g$) *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add: local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1/2 in exI*, *clarsimp*, *rule-tac x=1 in exI*)
by(*simp add: dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject*)

abbreviation *cnst-acc-flow* :: *real* \Rightarrow *real* \Rightarrow *real*[^]*program-vars* \Rightarrow *real*[^]*program-vars*
(φ_K)
where $\varphi_K\ a\ t\ s \equiv (\chi\ i. \text{if } i = (\downarrow_V\ "x") \text{ then } a * t^{\wedge} 2/2 + s\ \$\ (\downarrow_V\ "y") * t + s\ \$\ (\downarrow_V\ "x"))$
else $a * t + s\ \$\ (\downarrow_V\ "y")$)

lemma *local-flow-cnst-acc*: *local-flow* (*K g*) *UNIV UNIV* ($\varphi_K\ g$)
apply(*unfold-locales*, *simp-all add: local-lipschitz-def lipschitz-on-def*, *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 program-vars-univ-eq to-var-inject*)
apply(*clarsimp*, *case-tac i = \downarrow_V\ "x"*)
using *program-vars-exhaust* **by**(*auto intro!: poly-derivatives simp: to-var-inject*
vec-eq-iff)

lemma [*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 [*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 =  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$ 
  apply(subst Rat.sign-simps(18))+
  by(auto simp: 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*: $g < 0 \implies h \geq 0 \implies$

```

( $\lambda s. s|_V''x'' = h \wedge s|_V''y'' = 0$ )  $\leq$  fbox
(LOOP
  (( $x' = K g \ \& \ (\lambda s. s|_V''x'' \geq 0)$ ) ;
   (IF ( $\lambda s. s|_V''x'' = 0$ ) THEN ( $\downarrow_V''y'' ::= (\lambda s. - s|_V''y')$ ) ELSE skip))
  INV ( $\lambda s. s|_V''x'' \geq 0 \wedge 2 * g * s|_V''x'' = 2 * g * h + (s|_V''y'' * s|_V''y'')$ )
  ( $\lambda s. 0 \leq s|_V''x'' \wedge s|_V''x'' \leq h$ )
  apply(rule fbox-loopI, simp-all add: local-flow.fbox-g-ode[OF local-flow-cnst-acc])
  by (auto simp: bb-real-arith to-var-inject))

```

no-notation *cnst-acc-vec-field* (K)

```

and cnst-acc-flow ( $\varphi_K$ )
and to-var ( $\downarrow_V$ )
and val-p (infixl  $\downarrow_V$  90)

```

— Verified as a linear system (computing exponential).

abbreviation *cnst-acc-sq-mtx* :: $3 \text{ sq-mtx } (K)$

where $K \equiv \text{sq-mtx-chi } (\chi \ i :: 3. \text{ if } i=1 \text{ then } e \ 2 \text{ else if } i=2 \text{ then } e \ 3 \text{ else } 0)$

lemma *const-acc-mtx-pow2*: $K^2 = \text{sq-mtx-chi } (\chi \ i. \text{ if } i=1 \text{ then } e \ 3 \text{ else } 0)$

```

unfolding power2-eq-square times-sq-mtx-def
by(simp add: sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def)

```

lemma *const-acc-mtx-powN*: $n > 2 \implies (\tau *_R K)^n = 0$

```

apply(induct n, simp, case-tac  $n \leq 2$ )
apply(simp only: le-less-Suc-eq power-Suc, simp)
by(auto simp: const-acc-mtx-pow2 sq-mtx-chi-inject vec-eq-iff
  times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

```

lemma *exp-cnst-acc-sq-mtx*: $\exp(\tau *_R K) = ((\tau *_R K)^2 /_R 2) + (\tau *_R K) + 1$

unfolding exp-def **apply**(subst suminf-eq-sum[of 2])

using *const-acc-mtx-powN* **by** (*simp-all add: numeral-2-eq-2*)

lemma *exp-cnst-acc-eq*: $\exp (\tau *_R K) \$\$ i \$ j = \text{vector } (\text{map vector } ([1, \tau, \tau^2/2] \# [0, 1, \tau] \# [0, 0, 1] \# [])) \$ i \$ j$
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2 vector-def*
using *exhaust-3* **by** (*force simp: axis-def*)

lemma $\exp (\tau *_R K) \$\$ i \$ j = \text{vector } [1, \tau, \tau^2/2], \text{vector } [0, 1, \tau], \text{vector } [0, 0, 1] \$ i \$ j$
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2 vector-def*
using *exhaust-3* **by** (*force simp: axis-def*)

lemma *exp-cnst-acc-sq-mtx-simps*:
 $\exp (\tau *_R K) \$\$ 1 \$ 1 = 1 \exp (\tau *_R K) \$\$ 1 \$ 2 = \tau \exp (\tau *_R K) \$\$ 1 \$ 3 = \tau^2/2$
 $\exp (\tau *_R K) \$\$ 2 \$ 1 = 0 \exp (\tau *_R K) \$\$ 2 \$ 2 = 1 \exp (\tau *_R K) \$\$ 2 \$ 3 = \tau$
 $\exp (\tau *_R K) \$\$ 3 \$ 1 = 0 \exp (\tau *_R K) \$\$ 3 \$ 2 = 0 \exp (\tau *_R K) \$\$ 3 \$ 3 = 1$
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2*
by (*auto simp: plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def mat-def scaleR-vec-def axis-def plus-vec-def*)

lemma *bouncing-ball-sq-mtx*:
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 = h \wedge s\$2 = 0 \wedge 0 > s\$3) \leq \text{fbox}$
 $(\text{LOOP } ((x' = (*_V) K \ \& \ (\lambda s. s\$1 \geq 0))) ;$
 $(\text{IF } (\lambda s. s\$1 = 0) \text{ THEN } (2 ::= (\lambda s. - s\$2)) \text{ ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge s\$3 < 0 \wedge 2 * s\$3 * s\$1 = 2 * s\$3 * h + (s\$2 * s\$2)))$
 $(\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h)$
apply(*rule fbox-loopI*[*of -* ($\lambda s. 0 \leq s\$1 \wedge 0 > s\$3 \wedge 2 * s\$3 * s\$1 = 2 * s\$3 * h + (s\$2 * s\$2)$)])
apply(*force, force simp: bb-real-arith*)
apply(*simp add: local-flow.fbox-g-ode[OF local-flow-sq-mtx-linear] sq-mtx-vec-mult-eq*)
unfolding *UNIV-3* **apply**(*simp add: exp-cnst-acc-eq, safe*)
subgoal for *s t* **using** *bb-real-arith(2)*[*of s\\$3 s\\$1 h*] **by** (*force simp: field-simps*)
subgoal for *s t* **using** *bb-real-arith(3)*[*of s\\$3 s\\$1 h*] **by** (*simp add: field-simps*)
done

no-notation *cnst-acc-sq-mtx* (*K*)

Differential Ghosts

abbreviation *ghosts-vec-field* $s \equiv \chi \ i. \text{ if } i=1 \text{ then } - s\$1 \text{ else } 0$

abbreviation *ghosts-flow* $t \ s \equiv \chi \ i. \text{ if } i=1 \text{ then } s\$1 * \exp (- t) \text{ else } s\i

notation *ghosts-vec-field* (*f*)

and *ghosts-flow* (φ)

```
lemma ( $\lambda s :: \text{real}^2. s\$1 > 0$ )  $\leq$  fbox ( $x' = f \ \& \ (\lambda s. \text{True})$ ) ( $\lambda s. s\$1 > (0 :: \text{real})$ )
  apply (subst local-flow.fbox-g-ode [of f UNIV UNIV  $\varphi \ \lambda s. \text{True}$ ])
  apply (unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def)
  apply (clarsimp, rule-tac  $x=1$  in exI, clarsimp, rule-tac  $x=1$  in exI)
  apply (clarsimp simp: dist-norm norm-vec-def L2-set-def UNIV-2)
  unfolding real-sqrt-abs [symmetric] apply (rule real-le-lsqr, simp, simp)
  apply (smt power2-diff power2-sum real-less-rsqr zero-le-power2)
  by (auto simp: forall-2 vec-eq-iff intro!: poly-derivatives)
```

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

Thermostat

```
typedef thermostat-vars = {"t","T","on","TT"}
  morphisms to-str to-var
  apply (rule-tac  $x="t"$  in exI)
  by simp
```

notation to-var (\downarrow_V)

```
lemma number-of-thermostat-vars: CARD(thermostat-vars) = 4
  using type-definition.card type-definition-thermostat-vars by fastforce
```

```
instance thermostat-vars::finite
  apply (standard)
  apply (subst bij-betw-finite [of to-str UNIV {"t","T","on","TT"}])
  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 thermostat-vars-univ-eq:
  (UNIV::thermostat-vars set) = { $\downarrow_V "t", \downarrow_V "T", \downarrow_V "on", \downarrow_V "TT"$ }
  apply auto by (metis to-str to-str-inverse insertE singletonD)
```

```
lemma thermostat-vars-exhaust:  $x = \downarrow_V "t" \vee x = \downarrow_V "T" \vee x = \downarrow_V "on" \vee x = \downarrow_V "TT"$ 
  using thermostat-vars-univ-eq by auto
```

```
lemma thermostat-vars-sum:
  fixes  $f :: \text{thermostat-vars} \Rightarrow ('a::\text{banach})$ 
  shows  $(\sum (i::\text{thermostat-vars}) \in \text{UNIV}. f\ i) =$ 
 $f\ (\downarrow_V "t") + f\ (\downarrow_V "T") + f\ (\downarrow_V "on") + f\ (\downarrow_V "TT")$ 
  unfolding thermostat-vars-univ-eq by (simp add: to-var-inject)
```

abbreviation $\text{val-}T :: \text{real}^{\text{thermostat-vars}} \Rightarrow \text{string} \Rightarrow \text{real}$ (infixl \downarrow_V 90)

where $store \downarrow_V var \equiv store \$ \downarrow_V var$

lemma *thermostat-vars-allI*:

$P (\downarrow_V ''t'') \implies P (\downarrow_V ''T'') \implies P (\downarrow_V ''on'') \implies P (\downarrow_V ''TT'') \implies \forall i. P i$
 using *thermostat-vars-exhaust* by *metis*

abbreviation *temp-vec-field* :: $real \Rightarrow real \Rightarrow real \hat{=} thermostat\text{-}vars \Rightarrow real \hat{=} thermostat\text{-}vars$
 (f_T)

where $f_T a L s \equiv (\chi i. \text{if } i = \downarrow_V ''t'' \text{ then } 1 \text{ else } (\text{if } i = \downarrow_V ''T'' \text{ then } -a * (s \downarrow_V ''T'' - L) \text{ else } 0))$

abbreviation *temp-flow* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \hat{=} thermostat\text{-}vars \Rightarrow real \hat{=} thermostat\text{-}vars$
 (φ_T)

where $\varphi_T a L t s \equiv (\chi i. \text{if } i = \downarrow_V ''T'' \text{ then } -\exp(-a * t) * (L - s \downarrow_V ''T'') + L \text{ else } (\text{if } i = \downarrow_V ''t'' \text{ then } t + s \downarrow_V ''t'' \text{ else } (\text{if } i = \downarrow_V ''on'' \text{ then } s \downarrow_V ''on'' \text{ else } s \downarrow_V ''TT'')))$

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f_T a L s_1 - f_T a L s_2\| = |a| * |s_1 \downarrow_V ''T'' - s_2 \downarrow_V ''T''|$

proof(*simp add: norm-vec-def L2-set-def thermostat-vars-sum to-var-inject*)

assume *a1*: $0 < a$

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

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

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

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

hence $|a * (s_1 \downarrow_V ''T'' + -L) + - (a * (s_2 \downarrow_V ''T'' + -L))| = a * |s_1 \downarrow_V ''T'' - s_2 \downarrow_V ''T''|$

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

thus $|a * (s_2 \downarrow_V ''T'' - L) - a * (s_1 \downarrow_V ''T'' - L)| = a * |s_1 \downarrow_V ''T'' - s_2 \downarrow_V ''T''|$

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

qed

lemma *local-lipschitz-temp-dyn*:

assumes $0 < (a::real)$

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

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*)

apply(*unfold thermostat-vars-univ-eq, simp add: to-var-inject, clarsimp*)

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

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

apply(*unfold-locales, simp-all*)

using *local-lipschitz-temp-dyn* apply *blast*

apply(*rule thermostat-vars-allI, simp-all add: to-var-inject*)

using *thermostat-vars-exhaust* by (*auto intro!: poly-derivatives simp: vec-eq-iff to-var-inject*)

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

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

proof–

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

qed

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

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

proof–

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

qed

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

lemma *thermostat*:

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

no-notation *thermostat-vars.to-var* (\downarrow_V)
and *val-T* (**infixl** \downarrow_V 90)
and *temp-vec-field* (f_T)
and *temp-flow* (φ_T)

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

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

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

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

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

```

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

```

lemma *tank-arith*:

```

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

```

lemma *tank-flow*:

```

assumes  $0 \leq \tau$  and  $0 < c_o$  and  $c_o < c_i$ 
shows  $I \ hmin \ hmax \leq$ 
|LOOP
— control
((2 ::= (λs.0));(3 ::= (λs. s$1)));
(IF (λs. s$4 = 0 ∧ s$3 ≤ hmin + 1) THEN (4 ::= (λs.1)) ELSE
(IF (λs. s$4 = 1 ∧ s$3 ≥ hmax - 1) THEN (4 ::= (λs.0)) ELSE skip));
— dynamics
(IF (λs. s$4 = 0) THEN (x' = f (c_i - c_o) & G hmax (c_i - c_o) on {0..τ} UNIV
@ 0)
ELSE (x' = f (-c_o) & G hmin (-c_o) on {0..τ} 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* (φ)

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 *cat2funcset*

```

imports ../hs-prelims-dyn-sys 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 pred = 'a  $\Rightarrow$  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 fbF F S = {s. F s  $\subseteq$  S}
unfolding ffb-def map-dual-def klift-def kop-def dual-set-def
by(auto simp: Compl-eq-Diff-UNIV fun-eq-iff f2r-def converse-def r2f-def)

```

```

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

```

```

lemma ffb-iso: P  $\leq$  Q  $\implies$  fbF F P  $\leq$  fbF F Q
unfolding ffb-eq by auto

```

```

lemma ffb-invariants:
assumes {s. I s}  $\leq$  fbF F {s. I s} and {s. J s}  $\leq$  fbF F {s. J s}
shows {s. I s  $\wedge$  J s}  $\leq$  fbF F {s. I s  $\wedge$  J s}
and {s. I s  $\vee$  J s}  $\leq$  fbF 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]: fbF 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 ^ 'n)  $\Rightarrow$  'n  $\Rightarrow$  'a  $\Rightarrow$  'a ^ 'n
where vec-upd s i a = ( $\chi$  j. ((( $\$$ ) s)(i := a)) j)

```

```

definition assign :: 'n  $\Rightarrow$  ('a ^ 'n  $\Rightarrow$  'a)  $\Rightarrow$  ('a ^ 'n)  $\Rightarrow$  ('a ^ 'n) set (( $\mathcal{Q}$ - ::= -) [70,
65] 61)

```

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

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

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-def apply(simp add: kop-def klift-def map-dual-def)*
unfolding *dual-set-def f2r-def r2f-def* **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* :: $'a\ pred \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \Rightarrow 'b\ set)$
 $(IF - THEN - ELSE - [64, 64, 64]\ 63)$ **where**
 $IF\ P\ THEN\ X\ ELSE\ Y = (\lambda x. \text{if } P\ x\ \text{then } X\ x\ \text{else } Y\ x)$

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

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

We also deal with finite iteration.

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

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

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

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

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

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

0.5.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

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

lemma $\text{ffb-g-orbital-eq}: \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) Q = \{s. \forall X \in \text{Sols } (\lambda t. f) \ T \ S \ t_0 \ s. \forall t \in T. (\mathcal{P} \ X (\text{down } T \ t) \subseteq \{s. G \ s\}) \longrightarrow \mathcal{P} \ X (\text{down } T \ t) \subseteq Q\}$
unfolding ffb-g-orbital image-le-pred
apply $(\text{subgoal-tac } \forall X \ t. (\mathcal{P} \ X (\text{down } T \ t) \subseteq Q) = (\forall \tau \in \text{down } T \ t. (X \ \tau) \in Q))$
by $(\text{auto } \text{simp}: \text{image-def})$

context local-flow
begin

lemma $\text{ffb-g-ode}: \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ 0) Q = \{s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow (\varphi \ t \ s) \in Q)\} (\text{is} - = ?\text{wlp})$
unfolding ffb-g-orbital **apply** $(\text{safe}, \text{clarsimp})$

apply(*erule-tac* $x=\lambda t. \varphi \ t \ x$ **in** *ballE*)
using *in-ivp-sols* **apply**(*force*, *force*, *force simp: init-time ivp-sols-def*)
apply(*subgoal-tac* $\forall \tau \in \text{down } T \ t. X \ \tau = \varphi \ \tau \ x$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add: ivp-sols-def*)
using *init-time* **by** *auto*

lemma *ffb-g-ode-ivl*: $t \geq 0 \implies t \in T \implies \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } \{0..t\} \ S \ @ \ 0) \ Q$
 $=$

$\{s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \ \tau \ s)) \longrightarrow (\varphi \ t \ s) \in Q)\}$
unfolding *ffb-g-orbital* **apply**(*clarsimp*, *safe*)
apply(*erule-tac* $x=\lambda t. \varphi \ t \ x$ **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp: closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp: ivp-sols-def*)
apply(*subgoal-tac* $\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. X \ \tau = \varphi \ \tau \ x)$, *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add: ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp: closed-segment-eq-real-ivl*)
apply(*force simp: closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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* :: $((\text{'a}::\text{banach}) \Rightarrow \text{'a}) \Rightarrow \text{'a} \text{ pred} \Rightarrow \text{real set} \Rightarrow \text{'a set} \Rightarrow$
 $\text{real} \Rightarrow \text{'a pred} \Rightarrow (\text{'a} \Rightarrow \text{'a set}) ((1x' = - \ \& \ - \text{ on } - \ @ \ - \text{ DINV } -))$
where $(x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0 \text{ DINV } I) = (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0)$

lemma *ffb-g-orbital-guard*:
assumes $H = (\lambda s. G \ s \wedge Q \ s)$
shows $\text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \{s. Q \ s\} = \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \{s. H \ s\}$
unfolding *ffb-g-orbital* **using** *assms* **by** *auto*

lemma *ffb-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ I$ **and** $I \leq Q$
shows $P \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ Q$
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]*:
 $(\{s. I \ s\} \leq \text{fb}_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \{s. I \ s\}) = \text{diff-invariant } I \ f \ T \ S \ t_0 \ G$
by (*auto simp: diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *diff-invariant I f T S t_0 G* = $((\text{g-orbital } f \ G \ T \ S \ t_0)^\dagger) \ \{s. I \ s\} \subseteq \{s. I \ s\})$
unfolding *klift-def diff-invariant-def* **by** *simp*

lemma *bdf-diff-inv*:

diff-invariant $I f T S t_0 G = (bd_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T S @ t_0) \{s. I s\} \leq \{s. I s\})$
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 $\{s. I s\} \leq fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. True) \text{ on } T S @ t_0) \{s. I s\}$
shows $\{s. I s\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T S @ t_0) \{s. I s\}$
using *assms* **unfolding** *ffb-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*

begin

lemma *ffb-diff-inv-eq*: *diff-invariant* $I f T S 0 (\lambda s. True) =$

$(\{s. s \in S \longrightarrow I s\} = fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. True) \text{ on } T S @ 0) \{s. s \in S \longrightarrow I s\})$

unfolding *ffb-diff-inv[symmetric] ffb-g-orbital*
using *init-time* **apply**(*auto simp: subset-eq ivp-sols-def*)
apply(*subst ivp(2)[symmetric], simp*)
apply(*erule-tac x = \lambda t. \varphi t x in allE*)
using *in-domain has-vderiv-on-domain ivp(2) init-time* **by** *force*

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

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

end

lemma *ffb-g-odei*: $P \leq \{s. I s\} \Longrightarrow \{s. I s\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T S @ t_0) \{s. I s\} \Longrightarrow$

$\{s. I s \wedge G s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I) Q$
unfolding *g-ode-inv-def* **apply**(*rule-tac b = fb_{\mathcal{F}} (x' = f \ \& \ G \text{ on } T 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.

lemma *diff-solve-axiom*:

fixes *c::'a::\{heine-borel, banach\}*
assumes $0 \in T$ **and** *is-interval* $T \text{ open } T$
shows $fb_{\mathcal{F}} (x' = (\lambda s. c) \ \& \ G \text{ on } T \text{ UNIV } @ 0) Q =$
 $\{s. \forall t \in T. (\mathcal{P} (\lambda \tau. s + \tau *_R c) (\text{down } T t) \subseteq \{s. G s\}) \longrightarrow (s + t *_R c) \in Q\}$
apply(*subst local-flow.ffib-g-ode[of \lambda s. c - - (\lambda t s. s + t *_R c)]*)

using *line-is-local-flow* *assms* by *auto*

lemma *diff-solve-rule*:

assumes *local-flow* f T *UNIV* φ

and $\forall s. s \in P \longrightarrow (\forall t \in T. (\mathcal{P}(\lambda t. \varphi \ t \ s) \ (\text{down } T \ t) \subseteq \{s. G \ s\}) \longrightarrow (\varphi \ t \ s) \in Q)$

shows $P \leq \text{fb}_{\mathcal{F}}(x' = f \ \& \ G \ \text{on } T \ \text{UNIV} \ @ \ 0) \ Q$

using *assms* by(*subst local-flow.ffb-g-ode*) *auto*

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

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

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

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

assumes $\text{fb}_{\mathcal{F}}(x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ \{s. C \ s\} = \text{UNIV}$

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

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

proof

fix τ assume $\tau \in (\text{down } T \ t)$

hence $x \ \tau \in (x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ s$

using *assms*(2) by *blast*

also have $\forall y. y \in (x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ s \longrightarrow C \ y$

using *assms*(1) unfolding *ffb-eq* by *fastforce*

ultimately show $C \ (x \ \tau)$ by *blast*

qed

lemma *diff-cut-axiom*:

assumes *Thyp*: *is-interval* $T \ t_0 \in T$

and $\text{fb}_{\mathcal{F}}(x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ \{s. C \ s\} = \text{UNIV}$

shows $\text{fb}_{\mathcal{F}}(x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ Q = \text{fb}_{\mathcal{F}}(x' = f \ \& \ (\lambda s. G \ s \ \wedge \ C \ s) \ \text{on } T \ S \ @ \ t_0) \ Q$

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

fix s

{fix s' assume $s' \in (x' = f \ \& \ G \ \text{on } T \ S \ @ \ t_0) \ s$

then obtain $\tau :: \text{real}$ and X where *x-ivp*: $X \in \text{Sols}(\lambda t. f) \ T \ S \ t_0 \ s$

and $X \ \tau = s'$ and $\tau \in T$ and *guard-x*: $\mathcal{P} \ X \ (\text{down } T \ \tau) \subseteq \{s. G \ s\}$

using *g-orbitalD*[*of* $s' \ f \ G \ T \ S \ t_0 \ s$] by *blast*

have $\forall t \in (\text{down } T \ \tau). \mathcal{P} \ X \ (\text{down } T \ t) \subseteq \{s. G \ s\}$

using *guard-x* by (*force simp: image-def*)

also have $\forall t \in (\text{down } T \ \tau). t \in T$

using $\langle \tau \in T \rangle$ *Thyp closed-segment-subset-interval* by *auto*

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

using *g-orbitalI*[*OF* *x-ivp*] by (*metis (mono-tags, lifting)*)

hence $\forall t \in (\text{down } T \ \tau). C \ (X \ t)$

using *assms* unfolding *ffb-eq* by *fastforce*

hence $s' \in (x' = f \ \& \ (\lambda s. G \ s \ \wedge \ C \ s) \ \text{on } T \ S \ @ \ t_0) \ s$

using $g\text{-orbitalI}[OF\ x\text{-ivp}\ \langle\tau \in T\rangle]\ guard\text{-}x\ \langle X\ \tau = s'\rangle$ **by** $fastforce\}$
thus $(x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ s \subseteq (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ s$
by $blast$
next show $\bigwedge s. (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ s \subseteq (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ s$
by $(auto\ simp:\ g\text{-orbital}\text{-}eq)$
qed

lemma $diff\text{-}cut\text{-}rule$:

assumes $Thyp:\ is\text{-}interval\ T\ t_0 \in T$
and $ffb\text{-}C:\ P \leq fb_{\mathcal{F}}\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \{s. C\ s\}$
and $ffb\text{-}Q:\ P \leq fb_{\mathcal{F}}\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ Q$
shows $P \leq fb_{\mathcal{F}}\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ Q$
proof $(subst\ ffb\text{-}eq,\ subst\ g\text{-orbital}\text{-}eq,\ clarsimp)$
fix $t::real$ **and** $X::real \Rightarrow 'a$ **and** s **assume** $s \in P$ **and** $t \in T$
and $x\text{-ivp}:\ X \in Sols\ (\lambda t. f)\ T\ S\ t_0\ s$
and $guard\text{-}x:\ \forall \tau. s2p\ T\ \tau \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
have $\forall r \in (down\ T\ t). X\ r \in (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ s$
using $g\text{-orbitalI}[OF\ x\text{-ivp}]\ guard\text{-}x$ **by** $auto$
hence $\forall t \in (down\ T\ t). C\ (X\ t)$
using $ffb\text{-}C\ \langle s \in P \rangle$ **by** $(subst\ (asm)\ ffb\text{-}eq,\ auto)$
hence $X\ t \in (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ s$
using $guard\text{-}x\ \langle t \in T \rangle$ **by** $(auto\ intro!\ g\text{-orbitalI}\ x\text{-ivp})$
thus $(X\ t) \in Q$
using $\langle s \in P \rangle\ ffb\text{-}Q$ **by** $(subst\ (asm)\ ffb\text{-}eq)\ auto$
qed

The rules of dL

abbreviation $g\text{-global}\text{-}orbit :: (('a :: banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow 'a \Rightarrow 'a\ set$
 $((1x' = - \ \&\ -))\ where\ (x' = f \ \&\ G) \equiv (x' = f \ \&\ G\ on\ UNIV\ UNIV\ @\ 0)$

abbreviation $g\text{-global}\text{-}ode\text{-}inv :: (('a :: banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow 'a\ pred \Rightarrow 'a \Rightarrow 'a\ set$
 $((1x' = - \ \&\ -\ DINV\ -))\ where\ (x' = f \ \&\ G\ DINV\ I) \equiv (x' = f \ \&\ G\ on\ UNIV\ UNIV\ @\ 0\ DINV\ I)$

lemma $solve$:

assumes $local\text{-}flow\ f\ UNIV\ UNIV\ \varphi$
and $\forall s. s \in P \longrightarrow (\forall t. (\forall \tau \leq t. G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)$
shows $P \leq fb_{\mathcal{F}}\ (x' = f \ \&\ G)\ Q$
apply $(rule\ diff\text{-}solve\text{-}rule[OF\ assms(1)])$
using $assms(2)$ **by** $simp$

lemma DS :

fixes $c::'a::\{heine\text{-}borel,\ banach\}$
shows $fb_{\mathcal{F}}\ (x' = (\lambda s. c) \ \&\ G)\ Q = \{x. \forall t. (\forall \tau \leq t. G\ (x + \tau *_{\mathcal{R}} c)) \longrightarrow (x + t *_{\mathcal{R}} c) \in Q\}$
by $(subst\ diff\text{-}solve\text{-}axiom[of\ UNIV])\ auto$

lemma *DW*: $fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q = fb_{\mathcal{F}} (x' = f \ \& \ G) \ \{s. \ G \ s \longrightarrow s \in Q\}$
by (*rule diff-weak-axiom*)

lemma *dW*: $\{s. \ G \ s\} \leq Q \implies P \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q$
by (*rule diff-weak-rule*)

lemma *DC*:
assumes $fb_{\mathcal{F}} (x' = f \ \& \ G) \ \{s. \ C \ s\} = UNIV$
shows $fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q = fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s)) \ Q$
by (*rule diff-cut-axiom*) (*auto simp: assms*)

lemma *dC*:
assumes $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \ \{s. \ C \ s\}$
and $P \leq fb_{\mathcal{F}} (x' = f \ \& \ (\lambda s. \ G \ s \wedge C \ s)) \ Q$
shows $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q$
apply(*rule diff-cut-rule*)
using *assms* **by** *auto*

lemma *dI*:
assumes $P \leq \{s. \ I \ s\}$ **and** *diff-invariant I f UNIV UNIV 0 G* **and** $\{s. \ I \ s\} \leq Q$
shows $P \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \ Q$
by (*rule ffb-g-orbital-inv[OF assms(1) - assms(3)]*) (*simp add: assms(2)*)

end

0.5.4 Examples

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

theory *cat2funcset-examples*
imports *../hs-prelims-matrices cat2funcset*

begin

Preliminary lemmas for the examples.

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

lemma *four-eq-zero*: $(4::4) = 0$
by *simp*

lemma *UNIV-2*: $(UNIV::2 \text{ set}) = \{0, 1\}$
apply *safe using exhaust-2 two-eq-zero* **by** *auto*

lemma *UNIV-3*: $(UNIV::3 \text{ set}) = \{0, 1, 2\}$
apply *safe using exhaust-3 three-eq-zero* **by** *auto*

lemma *UNIV-4*: $(UNIV::4 \text{ set}) = \{0, 1, 2, 3\}$
apply *safe using exhaust-4 four-eq-zero* **by** *auto*

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 $s\$0$ to represent the x-coordinate and $s\$1$ for the y-coordinate. We prove that this motion remains circular.

— Verified with differential invariants.

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

lemma *pendulum-invariants*: $\{s. r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \{s. r^2 = (s\$0)^2 + (s\$1)^2\}$
by (*auto intro!*: *diff-invariant-rules poly-derivatives*)

— Verified with the flow.

abbreviation $pend-flow :: real \Rightarrow real^2 \Rightarrow real^2 (\varphi)$
where $\varphi t s \equiv (\chi i. \text{if } i = 0 \text{ then } s\$0 \cdot \cos t + s\$1 \cdot \sin t \text{ else } -s\$0 \cdot \sin t + s\$1 \cdot \cos t)$

lemma *local-flow-pend*: *local-flow f UNIV UNIV φ*
apply(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff, clarsimp*)
apply(*rule-tac x=1 in exI, clarsimp, rule-tac x=1 in exI*)
apply(*simp add: dist-norm norm-vec-def L2-set-def power2-commute UNIV-2*)
apply(*clarsimp, case-tac i = 0, simp*)
using *exhaust-2 two-eq-zero by (force intro!: poly-derivatives derivative-intros)+*

lemma *pendulum*: $\{s. r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}} (x' = f \ \& \ G) \{s. r^2 = (s\$0)^2 + (s\$1)^2\}$
by (*force simp: local-flow.ffb-g-ode[OF local-flow-pend]*)

— Verified by providing the dynamics

lemma *pendulum-dyn*: $\{s. r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}} (EVOL \ \varphi \ G \ T) \{s. r^2 = (s\$0)^2 + (s\$1)^2\}$
by *force*

— Verified as a linear system (using uniqueness).

abbreviation $pend-sq-mtx :: 2 \text{ sq-mtx } (A)$
where $A \equiv sq-mtx-chi (\chi i. \text{if } i=0 \text{ then } e \ 1 \text{ else } -e \ 0)$

lemma *pend-sq-mtx-exp-eq-flow*: $exp (t *_R A) *_V s = \varphi t s$
apply(*rule local-flow.eq-solution[OF local-flow-sq-mtx-linear, symmetric]*)
apply(*rule ivp-solsI, clarsimp*)
unfolding *sq-mtx-vec-mult-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro!: poly-derivatives simp: matrix-vector-mult-def*)

using *exhaust-2 two-eq-zero* **by** (*force simp: vec-eq-iff*, *auto*)

lemma *pendulum-sq-mtx*: $\{s. r^2 = (s\$0)^2 + (s\$1)^2\} \leq fb_{\mathcal{F}} (x' = (*_V) A \ \& \ G)$
 $\{s. r^2 = (s\$0)^2 + (s\$1)^2\}$

unfolding *local-flow.ffb-g-ode[OF local-flow-sq-mtx-linear]* *pend-sq-mtx-exp-eq-flow*
by *auto*

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

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 $s\$0$ to ball's height and $s\$1$ for its velocity. We prove that the ball remains above ground and below its initial resting position.

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:

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

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

proof—

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

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

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

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

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

by *auto*

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

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

ultimately have $h - x \geq 0$

by *linarith*

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

qed

abbreviation *fball* :: $real \Rightarrow real^2 \Rightarrow real^2$ (*f*)

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

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

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

(*LOOP* (

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


```

  (IF (λ s. s$0 = 0) THEN (1 ::= (λ s. - s$1)) ELSE skip))
  INV (λ s. 0 ≤ s$0 ∧ 2 · g · s$0 - 2 · g · h - s$1 · s$1 = 0))
  {s. 0 ≤ s$0 ∧ s$0 ≤ h}
  apply(rule ffb-loopI, simp-all)
  apply(force, force simp: bb-real-arith)
  apply(rule ffb-g-odei)
  by (auto intro!: diff-invariant-rules poly-derivatives simp: bb-real-arith)

```

— Verified with the flow.

abbreviation *ball-flow* :: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real}^2 \Rightarrow \text{real}^2$ (φ)
where $\varphi \ g \ t \ s \equiv (\chi \ i. \text{if } i=0 \text{ then } g \cdot t \wedge 2/2 + s\$1 \cdot t + s\$0 \text{ else } g \cdot t + s\$1)$

lemma *local-flow-ball*: *local-flow* (*f g*) UNIV UNIV ($\varphi \ g$)
apply(*unfold-locals*, *simp-all* *add*: *local-lipschitz-def* *lipschitz-on-def*, *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)
apply(*clarsimp*, *case-tac* $i = 0$)
using *exhaust-2* *two-eq-zero* **by** (*auto* *intro!*: *poly-derivatives* *simp*: *vec-eq-iff*)
force

lemma [*bb-real-arith*]:
assumes *invar*: $2 * g * x = 2 * g * h + v * v$
and *pos*: $g * \tau^2 / 2 + v * \tau + (x::\text{real}) = 0$
shows $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v)) = 0$
proof—
from *pos* **have** $g * \tau^2 + 2 * v * \tau + 2 * x = 0$ **by** *auto*
then **have** $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x = 0$
by (*metis* (*mono-tags*, *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 [*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* = $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
apply(*subst* *Rat.sign-simps*(18))+
by(*auto* *simp*: *semiring-normalization-rules*(29))
also **have** ... = $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$ (**is** ... = ?*middle*)

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

lemma *bouncing-ball*: $g < 0 \implies h \geq 0 \implies$
 $\{s. s\$0 = h \wedge s\$1 = 0\} \leq fb_{\mathcal{F}}$
 (LOOP (
 ($x' = (f\ g) \ \& \ (\lambda\ s. s\$0 \geq 0)$) ;
 (IF ($\lambda\ s. s\$0 = 0$) THEN ($1 ::= (\lambda s. - s\$1)$) ELSE skip))
 INV ($\lambda s. 0 \leq s\$0 \wedge 2 \cdot g \cdot s\$0 = 2 \cdot g \cdot h + s\$1 \cdot s\$1$)
 { $s. 0 \leq s\$0 \wedge s\$0 \leq h$ }
 by (rule *ffb-loopI*) (auto simp: *bb-real-arith local-flow.ffb-g-ode[OF local-flow-ball]*)
)

— Verified by providing the dynamics

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

— Verified as a linear system (computing exponential).

abbreviation *ball-sq-mtx* :: $3\ sq\text{-mtx}\ (A)$
 where *ball-sq-mtx* $\equiv sq\text{-mtx-chi}\ (\chi\ i. \text{if } i=0 \text{ then } e\ 1 \text{ else if } i=1 \text{ then } e\ 2 \text{ else } 0)$

lemma *ball-sq-mtx-pow2*: $A^2 = sq\text{-mtx-chi}\ (\chi\ i. \text{if } i=0 \text{ then } e\ 2 \text{ else } 0)$
 unfolding *power2-eq-square times-sq-mtx-def*
 by(*simp add: sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *ball-sq-mtx-powN*: $n > 2 \implies (\tau *_R A)^n = 0$
 apply(*induct n, simp, case-tac n ≤ 2*)
 apply(*simp only: le-less-Suc-eq power-Suc, simp*)
 by(*auto simp: ball-sq-mtx-pow2 sq-mtx-chi-inject vec-eq-iff*
times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

lemma *exp-ball-sq-mtx*: $\exp(\tau *_R A) = ((\tau *_R A)^2 /_R 2) + (\tau *_R A) + 1$
 unfolding *exp-def* apply(*subst suminf-eq-sum[of 2]*)
 using *ball-sq-mtx-powN* by (*simp-all add: numeral-2-eq-2*)

lemma *exp-ball-sq-mtx-simps*:

$\exp(\tau *_R A) \text{ \$\$ } 0 \text{ \$ } 0 = 1 \exp(\tau *_R A) \text{ \$\$ } 0 \text{ \$ } 1 = \tau \exp(\tau *_R A) \text{ \$\$ } 0 \text{ \$ } 2$
 $= \tau^2 / 2$
 $\exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 0 = 0 \exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 1 = 1 \exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 2$
 $= \tau$
 $\exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 0 = 0 \exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 1 = 0 \exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 2$
 $= 1$
unfolding *exp-ball-sq-mtx scaleR-power ball-sq-mtx-pow2*
by (*auto simp: plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def*
mat-def scaleR-vec-def axis-def plus-vec-def)

lemma *bouncing-ball-sq-mtx*:

$\{s. 0 \leq s\$0 \wedge s\$0 = h \wedge s\$1 = 0 \wedge 0 > s \$ 2\} \leq \text{fb}_{\mathcal{F}}$
 $(\text{LOOP } ((x' = (*_V) A \ \& \ (\lambda s. s\$0 \geq 0)) ;$
 $(\text{IF } (\lambda s. s\$0 = 0) \text{ THEN } (1 ::= (\lambda s. - s\$1)) \text{ ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot h + (s\$1 \cdot s\$1)))$
 $\{s. 0 \leq s\$0 \wedge s\$0 \leq h\}$
apply(*rule ffb-loopI, simp-all add: local-flow.ffib-g-ode[OF local-flow-sq-mtx-linear]*
sq-mtx-vec-mult-eq)
apply(*clarsimp, force simp: bb-real-arith*)
unfolding *UNIV-3* **apply**(*simp add: exp-ball-sq-mtx-simps, safe*)
using *bb-real-arith(2)* **apply**(*force simp: add commute mult commute*)
using *bb-real-arith(3)* **by** (*force simp: add commute mult commute*)

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

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 0 to denote the room's temperature, 1 is time as measured by the thermostat's chronometer, 2 is the temperature detected by the thermometer, and 3 states whether the heater is on ($s\$3 = 1$) or off ($s\$3 = 0$). 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 $f \ a \ L \ s \equiv (\chi \ i. \text{if } i = 1 \text{ then } 1 \text{ else } (\text{if } i = 0 \text{ then } -a * (s\$0 - L) \text{ else } 0))$

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

where $\varphi \ a \ L \ t \ s \equiv (\chi \ i. \text{if } i = 0 \text{ then } -\exp(-a * t) * (L - s\$0) + L \text{ else } (\text{if } i = 1 \text{ then } t + s\$1 \text{ else } (\text{if } i = 2 \text{ then } s\$2 \text{ else } s\$3)))$

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f\ a\ L\ s_1 - f\ a\ L\ s_2\| = |a| * |s_1\$0 - s_2\$0|$
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 $|a * (s_1\$0 + -\ L) + -\ (a * (s_2\$0 + -\ L))| = a * |s_1\$0 + -\ s_2\$0|$
using *a1* **by** (*simp add: abs-mult*)
thus $|a * (s_2\$0 - L) - a * (s_1\$0 - L)| = a * |s_1\$0 - s_2\$0|$
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 four-eq-zero)

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

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

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

lemmas *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 $\{s. Tmin \leq s\$0 \wedge s\$0 \leq Tmax \wedge s\$3 = 0\} \leq fb_{\mathcal{F}}$
(LOOP
 — *control*
 $((1 ::= (\lambda s. 0)); (2 ::= (\lambda s. s\$0)));$
 $(IF (\lambda s. s\$3 = 0 \wedge s\$2 \leq Tmin + 1) THEN (\mathfrak{I} ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$3 = 1 \wedge s\$2 \geq Tmax - 1) THEN (\mathfrak{I} ::= (\lambda s. 0)) ELSE skip));$
 — *dynamics*
 $(IF (\lambda s. s\$3 = 0) THEN (x' = (f a 0) \ \& \ (\lambda s. s\$1 \leq -(\ln(Tmin/s\$2))/a)$
on $\{0..t\}$ *UNIV @ 0)*
 $ELSE (x' = (f a L) \ \& \ (\lambda s. s\$1 \leq -(\ln((L - Tmax)/(L - s\$2))/a)$ *on* $\{0..t\}$
UNIV @ 0))
INV $(\lambda s. Tmin \leq s\$0 \wedge s\$0 \leq Tmax \wedge (s\$3 = 0 \vee s\$3 = 1))$
 $\{s. Tmin \leq s\$0 \wedge s\$0 \leq Tmax\}$
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* (φ)

end

0.6 Verification components with Kleene Algebras

We create verification rules based on various Kleene Algebras.

theory *hs-prelims-ka*
 imports
 KAT-and-DRA.PHL-KAT
 KAD.Modal-Kleene-Algebra
 Transformer-Semantics.Kleisli-Quantale

begin

0.6.1 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 (tt)

hide-const t

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

context *kat*

begin

— Definitions of Hoare Triple

definition *Hoare* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ (H) **where**
 $H\ p\ x\ q \longleftrightarrow \mathsf{tt}\ p \cdot x \leq x \cdot \mathsf{tt}\ q$

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

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

lemma *H-cons*: $\mathsf{tt}\ p \leq \mathsf{tt}\ p' \implies \mathsf{tt}\ q' \leq \mathsf{tt}\ q \implies H\ p'\ x\ q' \implies H\ p\ x\ q$
 by (*simp add: H-consl H-consr*)

— Skip program

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

— Sequential composition

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

— Conditional statement

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

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

proof —

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

— While loop

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

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

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

proof —

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

have $\text{tt } p \cdot n \ r = n \ r \cdot \text{tt } p \cdot n \ r$
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*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \cdot n \ r \leq \text{tt } q \implies H \ (\text{tt } i \cdot \text{tt } r) \ x \ i \implies H$
 $p \ (\text{while } r \ \text{inv } i \ \text{do } x \ \text{od}) \ q$
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* :: $'a \Rightarrow 'a \Rightarrow 'a \ (\text{loop} - \text{inv} - [64,64] \ 63)$
where $\text{loop } x \ \text{inv } i = x^*$

lemma *H-loop*: $H \ p \ x \ p \implies H \ p \ (\text{loop } x \ \text{inv } i) \ p$
unfolding *kat-loop-inv-def* **by** (*rule H-star*)

lemma *H-loop-inv*: $\text{tt } p \leq \text{tt } i \implies H \ i \ x \ i \implies \text{tt } i \leq \text{tt } q \implies H \ p \ (\text{loop } x \ \text{inv } i) \ q$
unfolding *kat-loop-inv-def* **using** *H-star-inv* **by** *blast*

— Invariants

lemma *H-inv*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies H \ i \ x \ i \implies H \ p \ x \ q$
by (*rule-tac p'=i and q'=i in H-cons*)

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

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

end

0.6.2 refinement KAT

```
class rkat = 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:  $H \ p \ x \ q \implies x \leq \text{Ref } p \ q$ 
  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 (Ref (tt v · tt p) q) else (Ref (n v · 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 (Ref (tt p · 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*: $\text{loop } (\text{Ref } p \ p) \ \text{inv } i \leq \text{Ref } p \ p$
unfolding *kat-loop-inv-def* **by** (*rule R-star*)

— Invariants

lemma *R-inv*: $\text{tt } p \leq \text{tt } i \implies \text{tt } i \leq \text{tt } q \implies \text{Ref } i \ i \leq \text{Ref } p \ q$
using *R-cons* **by** *force*

end

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)

0.6.3 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 $\text{if } p \text{ then } x \text{ else } y \text{ fi} = d \ p \cdot x + ad \ p \cdot y$

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*]: $| \text{if } p \text{ then } x \text{ else } y \text{ fi} | \ q = (ad \ p + |x| \ q) \cdot (d \ p + |y| \ q)$
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 $\text{loop } x \ \text{inv } i = x^*$

lemma *fbox-stari*: $d \ p \leq d \ i \implies d \ i \leq |x| \ i \implies d \ i \leq d \ q \implies d \ p \leq |x^*| \ q$
by (*meson* *dual-order.trans* *fbox-iso* *fbox-star-induct-var*)

lemma *fbox-loopi*: $d\ p \leq d\ i \implies d\ i \leq |x|\ i \implies d\ i \leq d\ q \implies d\ p \leq |loop\ x\ inv\ i|\ q$
unfolding *aka-loop-inv-def* **using** *fbox-stari* **by** *blast*

— Invariants

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

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

end

0.6.4 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*: $rel-ug.power\ X\ m = X^{\wedge m}$ **for** $X :: 'a\ rel$

proof (*induct m*)

case 0 **show** ?case

by (*metis rel-ug.power-0 relpow.simps(1)*)

case Suc **thus** ?case

by (*metis rel-ug.power-Suc2 relpow.simps(2)*)

qed

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

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

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

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

fix $x\ y\ z :: 'a\ rel$

show $Id \cup x\ O\ x^{\wedge *} \subseteq x^{\wedge *}$

by (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)

next

fix $x\ y\ z :: 'a\ rel$

```

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

```

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

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

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

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

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

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

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

0.6.5 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*: $(\bigwedge x. (f\bullet) \ x = (g\bullet) \ x) \Longrightarrow f = g$
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* $f = \text{qstar } f$ **for** $f :: 'a \text{ nd-fun}$

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** $x :: 'a \text{ nd-fun}$
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** $x :: 'a \text{ nd-fun}$
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** $x :: 'a \text{ nd-fun}$
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** $x :: 'a \text{ nd-fun}$
unfolding *less-eq-nd-fun-def less-nd-fun-def plus-nd-fun-def times-nd-fun-def sup-fun-def*
by (*unfold nd-fun-eq-iff le-fun-def, auto simp: kcomp-def*)

lemma *nd-star-one*[*nd-fun-aka*]: $1 + x \cdot x^\star \leq x^\star$
and *nd-star-unfoldl*[*nd-fun-aka*]: $z + x \cdot y \leq y \implies x^\star \cdot z \leq y$
and *nd-star-unfoldr*[*nd-fun-aka*]: $z + y \cdot x \leq y \implies z \cdot x^\star \leq y$ **for** $x :: 'a \text{ nd-fun}$
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]:  $n \ (n \ (1::'a \ nd\text{-fun})) = 1$ 
  and nd-fun-n-op-mult[nd-fun-aka]:  $n \ (n \ (n \ x \cdot n \ y)) = n \ x \cdot n \ y$ 
  and nd-fun-n-op-mult-comp[nd-fun-aka]:  $n \ x \cdot n \ (n \ x) = 0$ 
  and nd-fun-n-op-de-morgan[nd-fun-aka]:  $n \ (n \ (n \ x) \cdot n \ (n \ y)) = n \ x + n \ y$  for
x::'a 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  $\equiv (\lambda s. \bigcup \{(f \bullet) \ s \mid f. \text{Hoare } P \ f \ Q\})^\bullet$ 

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

end

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

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

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-fun
  unfolding antidomain-op-nd-fun-def times-nd-fun-def plus-nd-fun-def zero-nd-fun-def

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

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

end

end

```

0.7 Verification components with relational MKA

We show that relations form an antidomain Kleene algebra (hence a modal Kleene algebra). We use its forward box operator to derive rules in the algebra for weakest liberal preconditions (wlps) of hybrid programs. Finally, we derive our three methods for verifying correctness specifications for the continuous dynamics of HS in this setting.

theory *mka2rel*

imports *../hs-prelims-dyn-sys ../hs-prelims-ka*

begin

0.7.1 Store and weakest preconditions

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

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

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* (τ)

notation *Id* (*skip*)

and *zero-class.zero* (*0*)

and *rel-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-simps*[*simp*]:

$\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P\ s \longrightarrow Q\ s)$

$(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P\ s = Q\ s)$

$(\lceil P \rceil ; \lceil Q \rceil) = \lceil \lambda s. P\ s \wedge Q\ s \rceil$

$(\lceil P \rceil \cup \lceil Q \rceil) = \lceil \lambda s. P\ s \vee Q\ s \rceil$

rel-ad $\lceil P \rceil = \lceil \lambda s. \neg P\ s \rceil$

rel-aka.ads-d $\lceil P \rceil = \lceil P \rceil$

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

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

unfolding *rel-aka.fbox-def p2r-def rel-ad-def* **by** *auto*

definition *vec-upd* :: (*'a* \wedge *'b*) \Rightarrow *'b* \Rightarrow *'a* \Rightarrow *'a* \wedge *'b*

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

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

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

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

unfolding *wp-rel vec-upd-def assign-def* **by** (*auto simp: fun-upd-def*)

abbreviation *cond-sugar* :: $'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel\ (IF - THEN - ELSE - [64, 64]\ 63)$

where $IF\ P\ THEN\ X\ ELSE\ Y \equiv rel-aka.aka-cond\ \lceil P \rceil\ X\ Y$

abbreviation *loopi-sugar* :: $'a\ rel \Rightarrow 'a\ pred \Rightarrow 'a\ rel\ (LOOP - INV - [64, 64]\ 63)$

where $LOOP\ R\ INV\ I \equiv rel-aka.aka-loop-inv\ R\ \lceil I \rceil$

lemma *wp-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 *rel-aka.fbox-loopi*[*of* $\lceil P \rceil$] **by** *auto*

0.7.2 Verification of hybrid programs

Verification by providing evolution

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

where $EVOL\ \varphi\ G\ T = \{(s, s') \mid s\ s'.\ s' \in g-orbit\ (\lambda t. \varphi\ t\ s)\ G\ T\}$

lemma *wp-g-dyn*[*simp*]:

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

shows $wp\ (EVOL\ \varphi\ G\ T)\ \lceil Q \rceil = \lceil \lambda s. \forall t \in T. (\forall \tau \in down\ T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s) \rceil$

unfolding *wp-rel g-evol-def g-orbit-eq* **by** *auto*

Verification by providing solutions

definition *g-ode* :: $(('a :: banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ rel\ ((1x' = - \& - on - - @ -))$

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

lemma *wp-g-orbital*: $wp\ (x' = f \& G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil =$

$\lceil \lambda s. \forall X \in Sols\ (\lambda t. f)\ T\ S\ t_0\ s. \forall t \in T. (\forall \tau \in down\ T\ t. G\ (X\ \tau)) \longrightarrow Q\ (X\ t) \rceil$

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

context *local-flow*

begin

lemma *wp-g-ode*: $wp\ (x' = f \& G\ on\ T\ S\ @\ 0)\ \lceil Q \rceil =$

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

unfolding *wp-g-orbital apply*(*clarsimp, safe*)

apply(*erule-tac* $x = \lambda t. \varphi\ t\ s$ **in** *ballE*)

using *in-ivp-sols apply*(*force, force, force simp: init-time ivp-sols-def*)

apply(*subgoal-tac* $\forall \tau \in down\ T\ t. X\ \tau = \varphi\ \tau\ s$, *simp-all, clarsimp*)

apply(*subst eq-solution, simp-all add: ivp-sols-def*)

using *init-time* **by** *auto*

lemma *fbx-g-ode-ivl*: $t \geq 0 \implies t \in T \implies wp \ (x' = f \ \& \ G \text{ on } \{0..t\} \ S \ @ \ 0) \ [Q]$
 $=$
 $\lceil \lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)) \rceil$
unfolding *wp-g-orbital* **apply**(*clarsimp*, *safe*)
apply(*erule-tac* $x = \lambda t. \varphi \ t \ s$ **in** *ballE*, *force*)
using *in-ivp-sols-ivl* **apply**(*force simp: closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp: ivp-sols-def*)
apply(*subgoal-tac* $\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. X \ \tau = \varphi \ \tau \ s)$, *simp*, *clarsimp*)
apply(*subst eq-solution-ivl*, *simp-all add: ivp-sols-def*)
apply(*rule has-vderiv-on-subset*, *force*, *force simp: closed-segment-eq-real-ivl*)
apply(*force simp: closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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

end

Verification with differential invariants

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

lemma *wp-g-orbital-guard*:
assumes $H = (\lambda s. G \ s \wedge Q \ s)$
shows $wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [Q] = wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [H]$
unfolding *wp-g-orbital* **using** *assms* **by** *auto*

lemma *wp-g-orbital-inv*:
assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [I]$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [Q]$
using *assms(1)* **apply**(*rule order.trans*)
using *assms(2)* **apply**(*rule order.trans*)
apply(*rule rel-aka.fbx-iso*)
using *assms(3)* **by** *auto*

lemma *wp-diff-inv[simp]*: $(\lceil I \rceil \leq wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [I]) = \text{diff-invariant } I \ f \ T \ S \ t_0 \ G$
unfolding *diff-invariant-eq wp-g-orbital* **by**(*auto simp: p2r-def*)

lemma *diff-inv-guard-ignore*:
assumes $\lceil I \rceil \leq wp \ (x' = f \ \& \ (\lambda s. \text{True}) \text{ on } T \ S \ @ \ t_0) \ [I]$
shows $\lceil I \rceil \leq wp \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [I]$
using *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*

begin

lemma *wp-diff-inv-eq*: *diff-invariant* $I f T S 0 (\lambda s. \text{True}) =$
 $(\lceil \lambda s. s \in S \longrightarrow I s \rceil = \text{wp } (x' = f \ \& \ (\lambda s. \text{True}) \text{ on } T S @ 0) \lceil \lambda s. s \in S \longrightarrow I s \rceil)$
unfolding *wp-diff-inv*[*symmetric*] *wp-g-orbital*
using *init-time* **apply**(*clarsimp simp: ivp-sols-def*)
apply(*safe, force, force*)
apply(*subst ivp(2)[symmetric], simp*)
apply(*erule-tac x=\lambda t. \varphi t s in allE*)
using *in-domain has-vderiv-on-domain ivp(2) init-time* **by** *auto*

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

end

lemma *wp-g-odei*: $\lceil P \rceil \leq \lceil I \rceil \Longrightarrow \lceil I \rceil \leq \text{wp } (x' = f \ \& \ G \text{ on } T S @ t_0) \lceil I \rceil \Longrightarrow$
 $\lceil \lambda s. I s \wedge G s \rceil \leq \lceil Q \rceil \Longrightarrow$
 $\lceil P \rceil \leq \text{wp } (x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I) \lceil Q \rceil$
unfolding *g-ode-inv-def* **apply**(*rule-tac b=wp (x' = f \ \& \ G on T 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*)

0.7.3 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.

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

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

lemma *diff-weak-axiom*:

$wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil \lambda\ s.\ G\ s \longrightarrow Q\ s \rceil$

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

lemma *diff-weak-rule*:

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

shows $\lceil P \rceil \leq wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$

using *assms apply(subst wp-rel)*

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

lemma *wp-g-evol-IdD*:

assumes $wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = Id$

and $\forall \tau \in (down\ T\ t_0). (s, x\ \tau) \in (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)$

shows $\forall \tau \in (down\ T\ t_0). C\ (x\ \tau)$

proof

fix τ **assume** $\tau \in (down\ T\ t_0)$

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

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

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

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 *Thyp: is-interval* $T\ t_0 \in T$

and $wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = Id$

shows $wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$

proof *(rule-tac f = $\lambda x. wp\ x\ \lceil Q \rceil$ in HOL.arg-cong, rule subset-antisym)*

show $(x' = f \ \&\ G\ on\ T\ S\ @\ t_0) \subseteq (x' = f \ \&\ \lambda s. G\ s \wedge C\ s\ on\ T\ S\ @\ t_0)$

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

fix s **and** s' **assume** $s' \in g\text{-orbital}\ f\ G\ T\ S\ t_0\ s$

then obtain $\tau :: real$ **and** X **where** $x\text{-ivp}: X \in Sols\ (\lambda t. f)\ T\ S\ t_0\ s$

and $X\ \tau = s'$ **and** $\tau \in T$ **and** $guard\text{-}x: (\mathcal{P}\ X\ (down\ T\ \tau) \subseteq \{s. G\ s\})$

using *g-orbitalD[of s' f G T S t_0 s] by blast*

have $\forall t \in (down\ T\ \tau). \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s. G\ s\}$

using *guard-x by (force simp: image-def)*

also have $\forall t \in (down\ T\ \tau). t \in T$

using $\langle \tau \in T \rangle$ *Thyp by auto*

ultimately have $\forall t \in (down\ T\ \tau). X\ t \in g\text{-orbital}\ f\ G\ T\ S\ t_0\ s$

using *g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))*

hence $\forall t \in (down\ T\ \tau). C\ (X\ t)$

using *wp-g-evol-IdD[OF assms(3)] unfolding g-ode-def by blast*

thus $s' \in g\text{-orbital}\ f\ (\lambda s. G\ s \wedge C\ s)\ T\ S\ t_0\ s$

using *g-orbitalI[OF x-ivp] $\langle \tau \in T \rangle$ guard-x $\langle X\ \tau = s' \rangle$ by fastforce*

qed

next show $(x' = f \ \&\ \lambda s. G\ s \wedge C\ s\ on\ T\ S\ @\ t_0) \subseteq (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)$

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

qed

lemma *diff-cut-rule*:

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

The rules of dL

abbreviation *g-global-ode* :: $((a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel} \ ((1x' = - \ \& \ -))$
where $(x' = f \ \& \ G) \equiv (x' = f \ \& \ G \text{ on } UNIV \ UNIV \ @ \ 0)$

abbreviation *g-global-ode-inv* :: $((a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel}$
 $((1x' = - \ \& \ - \ DINV \ -))$ **where** $(x' = f \ \& \ G \ DINV \ I) \equiv (x' = f \ \& \ G \text{ on } UNIV \ UNIV \ @ \ 0 \ DINV \ I)$

lemma *DS*:

fixes $c::'a::\{heine-borel, banach\}$
shows $wp \ (x' = (\lambda s. \ c) \ \& \ G) \ \lceil Q \rceil = \lceil \lambda x. \ \forall t. \ (\forall \tau \leq t. \ G \ (x + \tau *_{\mathbb{R}} c)) \longrightarrow Q \ (x + t *_{\mathbb{R}} c) \rceil$
by (*subst diff-solve-axiom[of UNIV]*) *auto*

lemma *solve*:

assumes *local-flow* $f \ UNIV \ UNIV \ \varphi$
and $\forall s. \ P \ s \longrightarrow (\forall t. \ (\forall \tau \leq t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$
shows $\lceil P \rceil \leq wp \ (x' = f \ \& \ G) \ \lceil Q \rceil$
apply(*rule diff-solve-rule[OF assms(1)]*)
using *assms(2)* **by** *simp*

lemma *DW*: $wp \ (x' = f \ \& \ G) \ \lceil Q \rceil = wp \ (x' = f \ \& \ G) \ \lceil \lambda s. \ G \ s \longrightarrow Q \ s \rceil$
by (*rule diff-weak-axiom*)

lemma *dW*: $\lceil G \rceil \leq \lceil Q \rceil \Longrightarrow \lceil P \rceil \leq wp \ (x' = f \ \& \ G) \ \lceil Q \rceil$
by (*rule diff-weak-rule*)

lemma *DC*:

```

assumes  $wp\ (x' = f \ \&\ G)\ \lceil C \rceil = Id$ 
shows  $wp\ (x' = f \ \&\ G)\ \lceil Q \rceil = wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s))\ \lceil Q \rceil$ 
apply (rule diff-cut-axiom)
using assms by auto

```

lemma *dC*:

```

assumes  $\lceil P \rceil \leq wp\ (x' = f \ \&\ G)\ \lceil C \rceil$ 
and  $\lceil P \rceil \leq wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s))\ \lceil Q \rceil$ 
shows  $\lceil P \rceil \leq wp\ (x' = f \ \&\ G)\ \lceil Q \rceil$ 
apply(rule diff-cut-rule)
using assms by auto

```

lemma *dI*:

```

assumes  $\lceil P \rceil \leq \lceil I \rceil$  and diff-invariant I f UNIV UNIV 0 G and  $\lceil I \rceil \leq \lceil Q \rceil$ 
shows  $\lceil P \rceil \leq wp\ (x' = f \ \&\ G)\ \lceil Q \rceil$ 
apply(rule wp-g-orbital-inv[OF assms(1) - assms(3)])
unfolding wp-diff-inv using assms(2) .

```

end

0.8 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 *mka2ndfun*

imports

../hs-prelims-dyn-sys

../hs-prelims-ka

begin

0.8.1 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*

notation *fbox* (*wp*)

no-notation *bqtran* ($\lfloor \cdot \rfloor$)
and *Archimedean-Field.ceiling* ($\lceil \cdot \rceil$)
and *Archimedean-Field.floor* ($\lfloor \cdot \rfloor$)
and *Relation.relcomp* (**infixl** ; 75)
and *Range-Semiring.antirange-semiring-class.ars-r* (r)
and *antidomain-semiringl.ads-d* (d)
and *Hoare* (H)
and *n-op* (n - [90] 91)
and *tau* (τ)

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

lemma *p2ndf-simps*[*simp*]:
 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P\ s \longrightarrow Q\ s)$
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P\ s = Q\ s)$
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P\ s \wedge Q\ s \rceil$
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P\ s \vee Q\ s \rceil$
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-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*)

lemma *wp-nd-fun2*: $wp\ (F_\bullet)\ \lceil P \rceil = \lceil \lambda s. \forall s'. s' \in (F\ s) \longrightarrow P\ s' \rceil$
by (*subst wp-nd-fun, simp*)

abbreviation *ndf2p* :: 'a *nd-fun* \Rightarrow 'a \Rightarrow *bool* ($(1 \lfloor \cdot \rfloor)$)
where $\lfloor f \rfloor \equiv (\lambda x. x \in \text{Domain}\ (\mathcal{R}\ (f_\bullet)))$

lemma *p2ndf-ndf2p-id*: $F \leq \eta^\bullet \Longrightarrow \lceil \lfloor F \rfloor \rceil = F$
unfolding *f2r-def* **apply**(*rule nd-fun-ext*)
apply(*subgoal-tac* $\forall x. (F_\bullet)\ x \subseteq \{x\}$, *simp*)
by(*blast, simp add: le-fun-def less-eq-nd-fun.rep-eq*)

lemma *p2ndf-ndf2p-wp*: $\lceil \lfloor wp\ R\ P \rfloor \rceil = wp\ R\ P$
apply(*rule p2ndf-ndf2p-id*)
by (*simp add: a-subid fbox-def one-nd-fun.transfer*)

lemma *ndf2p-wpD*: $\lfloor wp\ F\ \lceil Q \rceil \rfloor\ s = (\forall s'. s' \in (F_\bullet)\ s \longrightarrow Q\ s')$
apply(*subgoal-tac* $F = (F_\bullet)^\bullet$)
apply(*rule ssubst[of F (F_\bullet)^\bullet]*, *simp*)
apply(*subst wp-nd-fun*)
by(*simp-all add: f2r-def*)

We check that *wp* coincides with our other definition of the forward box

operator $fb_{\mathcal{F}} = \partial_F \circ bd_{\mathcal{F}} \circ op_K$.

lemma *ffb-is-wp*: $fb_{\mathcal{F}} (F \bullet) \{x. P \ x\} = \{s. \lfloor wp \ F \ \lceil P \rceil \rfloor \ s\}$
unfolding *ffb-def* **unfolding** *map-dual-def* *klift-def* *kop-def* *fbox-def*
unfolding *r2f-def* *f2r-def* **apply** *clarsimp*
unfolding *antidomain-op-nd-fun-def* **unfolding** *dual-set-def*
unfolding *times-nd-fun-def* *kcomp-def* **by** *force*

lemma *wp-is-ffb*: $wp \ F \ P = (\lambda x. \{x\} \cap fb_{\mathcal{F}} (F \bullet) \{s. \lfloor P \rfloor \ s\})^\bullet$
apply(*rule nd-fun-ext*, *simp*)
unfolding *ffb-def* **unfolding** *map-dual-def* *klift-def* *kop-def* *fbox-def*
unfolding *r2f-def* *f2r-def* **apply** *clarsimp*
unfolding *antidomain-op-nd-fun-def* **unfolding** *dual-set-def*
unfolding *times-nd-fun-def* **apply** *auto*
unfolding *kcomp-prop* **by** *auto*

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

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

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

lemma *wp-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*

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

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

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

lemma *wp-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*

0.8.2 Verification of hybrid programs

Verification by providing evolution

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

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

Verification by providing solutions

definition *g-ode* :: $((a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow$
 $real \Rightarrow 'a\ nd-fun ((1x' = - \& -\ on - - @ -))$
where $(x' = f \& G\ on\ T\ S @ t_0) \equiv (\lambda s. g-orbital\ f\ G\ T\ S\ t_0\ s)^\bullet$

lemma *wp-g-orbital*: $wp (x' = f \& G\ on\ T\ S @ t_0) \lceil Q \rceil =$
 $\lceil \lambda s. \forall X \in ivp-sols\ (\lambda t. f)\ T\ S\ t_0\ s. \forall t \in T. (\forall \tau \in down\ T\ t. G (X\ \tau)) \longrightarrow Q (X\ t) \rceil$
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*: $wp (x' = f \& G\ on\ T\ S @ 0) \lceil Q \rceil =$
 $\lceil \lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in down\ T\ t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \rceil$
unfolding *wp-g-orbital* **apply**(*clarsimp, safe*)
apply(*erule-tac x = \lambda t. \varphi t s in ballE*)
using *in-ivp-sols* **apply**(*force, force, force simp: init-time ivp-sols-def*)
apply(*subgoal-tac \forall \tau \in down\ T\ t. X \tau = \varphi \tau s, simp-all, clarsimp*)
apply(*subst eq-solution, simp-all add: ivp-sols-def*)
using *init-time* **by** *auto*

lemma *fbox-g-ode-ivl*: $t \geq 0 \implies t \in T \implies wp (x' = f \& G\ on\ \{0..t\}\ S @ 0) \lceil Q \rceil$
 $=$
 $\lceil \lambda s. s \in S \longrightarrow (\forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \rceil$
unfolding *wp-g-orbital* **apply**(*clarsimp, safe*)
apply(*erule-tac x = \lambda t. \varphi t s in ballE, force*)
using *in-ivp-sols-ivl* **apply**(*force simp: closed-segment-eq-real-ivl*)
using *in-ivp-sols-ivl* **apply**(*force simp: ivp-sols-def*)
apply(*subgoal-tac \forall t \in \{0..t\}. (\forall \tau \in \{0..t\}. X \tau = \varphi \tau s), simp, clarsimp*)
apply(*subst eq-solution-ivl, simp-all add: ivp-sols-def*)
apply(*rule has-vderiv-on-subset, force, force simp: closed-segment-eq-real-ivl*)
apply(*force simp: closed-segment-eq-real-ivl*)
using *interval-time init-time* **apply** (*meson is-interval-1 order-trans*)
using *init-time* **by** *force*

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* :: $((a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow$

real \Rightarrow 'a *pred* \Rightarrow 'a *nd-fun* ((*1x'* = - & - on - - @ - *DINV* -))
where (*x'* = *f* & *G* on *T S* @ *t*₀ *DINV I*) = (*x'* = *f* & *G* on *T S* @ *t*₀)

lemma *wp-g-orbital-guard*:

assumes *H* = ($\lambda s. G\ s \wedge Q\ s$)
shows *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*Q*] = *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*H*]
unfolding *wp-g-orbital* **using** *assms* **by** *auto*

lemma *wp-g-orbital-inv*:

assumes [*P*] \leq [*I*] **and** [*I*] \leq *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*I*] **and** [*I*] \leq [*Q*]
shows [*P*] \leq *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*Q*]
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]*: ([*I*] \leq *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*I*]) = *diff-invariant* *I f T S t*₀ *G*

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

lemma *diff-inv-guard-ignore*:

assumes [*I*] \leq *wp* (*x'* = *f* & ($\lambda s. \text{True}$) on *T S* @ *t*₀) [*I*]
shows [*I*] \leq *wp* (*x'* = *f* & *G* on *T S* @ *t*₀) [*I*]
using *assms* **unfolding** *wp-diff-inv diff-invariant-eq* **by** *auto*

context *local-flow*

begin

lemma *wp-diff-inv-eq: diff-invariant I f T S 0* ($\lambda s. \text{True}$) =

($[\lambda s. s \in S \longrightarrow I\ s] = \text{wp} (x' = f \ \& \ (\lambda s. \text{True}) \text{ on } T\ S \ @ \ 0) [\lambda s. s \in S \longrightarrow I\ s]$)

unfolding *wp-diff-inv[symmetric]* *wp-g-orbital*
using *init-time* **apply**(*clarsimp simp: ivp-sols-def*)
apply(*safe, force, force*)
apply(*subst ivp(2)[symmetric], simp*)
apply(*erule-tac x = \lambda t. \varphi\ t\ s in allE*)
using *in-domain has-vderiv-on-domain ivp(2) init-time* **by** *auto*

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

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

end

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

[*P*] \leq *wp* (*x'* = *f* & *G* on *T S* @ *t*₀ *DINV I*) [*Q*]
unfolding *g-ode-inv-def* **apply**(*rule-tac b = wp (x' = f & G on T S @ t0) [I]*) **in**

order.trans)
apply(*rule-tac* $I=I$ **in** *wp-g-orbital-inv*, *simp-all*)
apply(*subst wp-g-orbital-guard*, *simp*)
by (*rule fbox-iso*, *simp*)

0.8.3 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.

lemma *diff-solve-axiom*:

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

lemma *diff-solve-rule*:

assumes *local-flow* $f \ T \ \text{UNIV} \ \varphi$
and $\forall s. P \ s \longrightarrow (\forall t \in T. (\mathcal{P} (\lambda t. \varphi \ t \ s) (\text{down } T \ t) \subseteq \{s. G \ s\}) \longrightarrow Q (\varphi \ t \ s))$
shows $[P] \leq \text{wp } (x' = f \ \& \ G \text{ on } T \text{ UNIV } @ \ 0) \ [Q]$
using *assms* **by**(*subst local-flow.wp-g-ode*, *auto*)

lemma *diff-weak-axiom*:

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

lemma *diff-weak-rule*: $[G] \leq [Q] \implies [P] \leq \text{wp } (x' = f \ \& \ G \text{ on } T \ S @ \ t_0) \ [Q]$
by (*subst wp-g-orbital*) (*auto simp: g-ode-def*)

lemma *wp-g-orbit-IdD*:

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

proof

fix τ **assume** $\tau \in (\text{down } T \ t)$
hence $x \ \tau \in g\text{-orbital } f \ G \ T \ S \ t_0 \ s$
using *assms*(2) **by** *blast*
also have $\forall y. y \in (g\text{-orbital } f \ G \ T \ S \ t_0 \ s) \longrightarrow C \ y$
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 Thyp: is-interval  $T\ t_0 \in T$ 
and  $wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ [C] = \eta^\bullet$ 
shows  $wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ [Q] = wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ [Q]$ 
proof(rule-tac  $f = \lambda x. wp\ x\ [Q]$  in HOL.arg-cong, rule-nd-fun-ext, rule-subset-antisym)
  fix  $s$  show  $((x' = f \ \&\ G\ on\ T\ S\ @\ t_0) \bullet) s \subseteq ((x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0) \bullet) s$ 
  proof(clarsimp simp: g-ode-def)
    fix  $s'$  assume  $s' \in g\text{-orbital}\ f\ G\ T\ S\ t_0\ s$ 
    then obtain  $\tau :: real$  and  $X$  where  $x\text{-ivp}: X \in ivp\text{-sols}\ (\lambda t. f)\ T\ S\ t_0\ s$ 
    and  $X\ \tau = s'$  and  $\tau \in T$  and  $guard\text{-}x: (\mathcal{P}\ X\ (down\ T\ \tau) \subseteq \{s. G\ s\})$ 
    using g-orbitalD[of s' f G T S t_0 s] by blast
    have  $\forall t \in (down\ T\ \tau). \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s. G\ s\}$ 
    using guard-x by (force simp: image-def)
    also have  $\forall t \in (down\ T\ \tau). t \in T$ 
    using  $\langle \tau \in T \rangle$  Thyp by auto
    ultimately have  $\forall t \in (down\ T\ \tau). X\ t \in g\text{-orbital}\ f\ G\ T\ S\ t_0\ s$ 
    using g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))
    hence  $\forall t \in (down\ T\ \tau). C\ (X\ t)$ 
    using wp-g-orbit-IdD[OF assms(3)] by blast
    thus  $s' \in g\text{-orbital}\ f\ (\lambda s. G\ s \wedge C\ s)\ T\ S\ t_0\ s$ 
    using g-orbitalI[OF x-ivp <tau> T] guard-x  $\langle X\ \tau = s' \rangle$  by fastforce
  qed
next
  fix  $s$  show  $((x' = f \ \&\ \lambda s. G\ s \wedge C\ s\ on\ T\ S\ @\ t_0) \bullet) s \subseteq ((x' = f \ \&\ G\ on\ T\ S\ @\ t_0) \bullet) s$ 
  by (auto simp: g-orbital-eq g-ode-def)
qed

lemma diff-cut-rule:
  assumes Thyp: is-interval  $T\ t_0 \in T$ 
  and  $wp\text{-}C: [P] \leq wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ [C]$ 
  and  $wp\text{-}Q: [P] \leq wp\ (x' = f \ \&\ (\lambda s. G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ [Q]$ 
  shows  $[P] \leq wp\ (x' = f \ \&\ G\ on\ T\ S\ @\ t_0)\ [Q]$ 
proof(simp add: wp-nd-fun g-orbital-eq g-ode-def, clarsimp)
  fix  $t :: real$  and  $X :: real \Rightarrow 'a$  and  $s$  assume  $P\ s$  and  $t \in T$ 
  and  $x\text{-ivp}: X \in ivp\text{-sols}\ (\lambda t. f)\ T\ S\ t_0\ s$ 
  and  $guard\text{-}x: \forall x. x \in T \wedge x \leq t \longrightarrow G\ (X\ x)$ 
  have  $\forall t \in (down\ T\ t). X\ t \in g\text{-orbital}\ f\ G\ T\ S\ t_0\ s$ 
  using g-orbitalI[OF x-ivp] guard-x by auto
  hence  $\forall t \in (down\ T\ t). C\ (X\ t)$ 
  using wp-C  $\langle P\ s \rangle$  by (subst (asm) wp-nd-fun, auto simp: g-ode-def)
  hence  $X\ t \in g\text{-orbital}\ f\ (\lambda s. G\ s \wedge C\ s)\ T\ S\ t_0\ s$ 
  using guard-x  $\langle t \in T \rangle$  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

```

The rules of dL

abbreviation *g-global-ode* :: (('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a nd-fun ((1x'=- & -))

where (x' = f & G) ≡ (x' = f & G on UNIV UNIV @ 0)

abbreviation *g-global-ode-inv* :: (('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a pred ⇒ 'a nd-fun

((1x'=- & - DINV I)) **where** (x' = f & G DINV I) ≡ (x' = f & G on UNIV UNIV @ 0 DINV I)

lemma *DS*:

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

shows wp (x'=(λs. c) & G) [Q] = [λx. ∀t. (∀τ≤t. G (x + τ *_R c)) → Q (x + t *_R c)]

by (subst diff-solve-axiom[of UNIV]) (auto simp: fun-eq-iff)

lemma *solve*:

assumes local-flow f UNIV UNIV φ

and ∀s. P s → (∀t. (∀τ≤t. G (φ τ s)) → Q (φ t s))

shows [P] ≤ wp (x' = f & G) [Q]

apply(rule diff-solve-rule[OF assms(1)])

using assms(2) **by** simp

lemma *DW*: wp (x' = f & G) [Q] = wp (x' = f & G) [λs. G s → Q s]

by (rule diff-weak-axiom)

lemma *dW*: [G] ≤ [Q] ⇒ [P] ≤ wp (x' = f & G) [Q]

by (rule diff-weak-rule)

lemma *DC*:

assumes wp (x' = f & G) [C] = η•

shows wp (x' = f & G) [Q] = wp (x' = f & (λs. G s ∧ C s)) [Q]

apply (rule diff-cut-axiom)

using assms **by** auto

lemma *dC*:

assumes [P] ≤ wp (x' = f & G) [C]

and [P] ≤ wp (x' = f & (λs. G s ∧ C s)) [Q]

shows [P] ≤ wp (x' = f & G) [Q]

apply(rule diff-cut-rule)

using assms **by** auto

lemma *dI*:

assumes [P] ≤ [I] **and** diff-invariant I f UNIV UNIV 0 G **and** [I] ≤ [Q]

shows [P] ≤ wp (x' = f & G) [Q]

apply(rule wp-g-orbital-inv[OF assms(1) - assms(3)])

unfolding wp-diff-inv **using** assms(2) .

end

0.8.4 Examples

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

```
theory mka-examples
imports ../hs-prelims-matrices mka2rel
```

begin

Preliminary preparation for the examples.

```
no-notation Archimedean-Field.ceiling ( $\lceil \cdot \rceil$ )
and Archimedean-Field.floor-ceiling-class.floor ( $\lfloor \cdot \rfloor$ )
```

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[OF local-flow-pend]*)

— Verified as a linear system (using uniqueness).

abbreviation *pend-sq-mtx* :: $2 \text{ sq-mtx } (A)$
where $A \equiv \text{sq-mtx-chi } (\chi \ i. \text{ if } i=1 \text{ then } e \ 2 \text{ else } - \ e \ 1)$

lemma *pend-sq-mtx-exp-eq-flow*: $\exp \ (t *_{\mathcal{R}} A) *_{\mathcal{V}} s = \varphi \ t \ s$
apply(*rule local-flow.eq-solution[OF local-flow-sq-mtx-linear, symmetric]*)
apply(*rule ivp-solsI, simp add: sq-mtx-vec-mult-def matrix-vector-mult-def*)
apply(*force intro!: poly-derivatives simp: matrix-vector-mult-def*)
using *exhaust-2* **by** (*force simp: vec-eq-iff, auto*)

lemma *pendulum-sq-mtx*:
 $\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil \leq wp \ (x' = ((*_{\mathcal{V}}) A) \ \& \ G) \ \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
unfolding *local-flow.wp-g-ode[OF local-flow-sq-mtx-linear]* *pend-sq-mtx-exp-eq-flow*
by *auto*

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

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 (0 > g) by auto
  hence obs: v * v = 2 * g * (x - h) ∧ 0 > g ∧ v * v ≥ 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) ≤ 0
    using divide-nonneg-neg by fastforce
  ultimately have h - x ≥ 0
    by linarith
  thus ?thesis by auto
qed

```

lemma *bouncing-ball-inv*:

```

  fixes h::real
  shows g < 0 ⟹ h ≥ 0 ⟹ [λs. s$1 = h ∧ s$2 = 0] ≤
    wp
      (LOOP
        ((x' = f g & (λ s. s$1 ≥ 0) DINV (λs. 2 * g * s$1 - 2 * g * h - s$2 *
s$2 = 0));
        (IF (λ s. s$1 = 0) THEN (2 ::= (λs. - s$2)) ELSE skip))
        INV (λs. 0 ≤ s$1 ∧ 2 * g * s$1 - 2 * g * h - s$2 * s$2 = 0)
      ) [λs. 0 ≤ s$1 ∧ s$1 ≤ h]
  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 * τ2 / 2 + v * τ + (x::real) = 0
  shows 2 * g * h + (g * τ * (g * τ + v) + v * (g * τ + v)) = 0
proof-
  from pos have g * τ2 + 2 * v * τ + 2 * x = 0 by auto
  then have g2 * τ2 + 2 * g * v * τ + 2 * g * x = 0
    by (metis (mono-tags, hide-lams) Groups.mult-ac(1,3) mult-zero-right
      monoid-mult-class.power2-eq-square semiring-class.distrib-left)
  hence g2 * τ2 + 2 * g * v * τ + v2 + 2 * g * h = 0
    using invar by (simp add: monoid-mult-class.power2-eq-square)
  hence obs: (g * τ + v)2 + 2 * g * h = 0
    apply (subst power2-sum) by (metis (no-types, 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 * τ * (g * τ + v) + v * (g * τ + 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 * τ2 / 2 + v * τ + (x::real)) =

```



```

 $2 * g * h + (g * \tau * (g * \tau + v) + v * (g * \tau + v))$  (is ?lhs = ?rhs)
proof–
  have ?lhs =  $g^2 * \tau^2 + 2 * g * v * \tau + 2 * g * x$ 
    apply(subst Rat.sign-simps(18))+
    by(auto simp: 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  $g < 0 \implies h \geq 0 \implies$ 
     $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$ 
    (LOOP
      ((EVOL ( $\varphi$  g) ( $\lambda s. 0 \leq s\$1$ ) T);
      (IF ( $\lambda s. s\$1 = 0$ ) THEN ( $2 ::= (\lambda s. - s\$2)$ ) ELSE skip))
      INV ( $\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$ ))
     $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$ 
  by (rule wp-loopI) (auto simp: bb-real-arith)

```

— Verified with the flow.

```

lemma local-flow-ball: local-flow (f g) UNIV UNIV ( $\varphi$  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  $g < 0 \implies h \geq 0 \implies$ 
     $[\lambda s. s\$1 = h \wedge s\$2 = 0] \leq wp$ 
    (LOOP
      (( $x' = f$  g & ( $\lambda s. s\$1 \geq 0$ ));
      (IF ( $\lambda s. s\$1 = 0$ ) THEN ( $2 ::= (\lambda s. - s\$2)$ ) ELSE skip))
      INV ( $\lambda s. 0 \leq s\$1 \wedge 2 * g * s\$1 = 2 * g * h + s\$2 * s\$2$ ))
     $[\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$ 
  apply(rule wp-loopI, simp-all add: local-flow.wp-g-ode[OF local-flow-ball])

```

by (*auto simp: bb-real-arith*)

— Verified as a linear system (computing exponential).

abbreviation *ball-sq-mtx* :: \mathcal{I} *sq-mtx* (*A*)

where *ball-sq-mtx* \equiv *sq-mtx-chi* (χ *i. if i = 1 then e 2 else if i = 2 then e 3 else 0*)

lemma *ball-sq-mtx-pow2*: $A^2 = \text{sq-mtx-chi } (\chi \text{ i. if i = 1 then e 3 else 0})$

unfolding *monoid-mult-class.power2-eq-square times-sq-mtx-def*

by (*simp add: sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *ball-sq-mtx-powN*: $n > 2 \implies (\tau *_R A)^{\wedge n} = 0$

apply(*induct n, simp, case-tac n ≤ 2*)

apply(*simp only: le-less-Suc-eq power-class.power.simps(2), simp*)

by (*auto simp: ball-sq-mtx-pow2 sq-mtx-chi-inject vec-eq-iff times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def*)

lemma *exp-ball-sq-mtx*: $\exp(\tau *_R A) = ((\tau *_R A)^2 /_R 2) + (\tau *_R A) + 1$

unfolding *exp-def* **apply**(*subst suminf-eq-sum[of 2]*)

using *ball-sq-mtx-powN* **by** (*simp-all add: numeral-2-eq-2*)

lemma *exp-ball-sq-mtx-simps*:

$\exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 1 = 1 \exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 2 = \tau \exp(\tau *_R A) \text{ \$\$ } 1 \text{ \$ } 3 = \tau^{\wedge 2/2}$

$\exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 1 = 0 \exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 2 = 1 \exp(\tau *_R A) \text{ \$\$ } 2 \text{ \$ } 3 = \tau$

$\exp(\tau *_R A) \text{ \$\$ } 3 \text{ \$ } 1 = 0 \exp(\tau *_R A) \text{ \$\$ } 3 \text{ \$ } 2 = 0 \exp(\tau *_R A) \text{ \$\$ } 3 \text{ \$ } 3 = 1$

unfolding *exp-ball-sq-mtx scaleR-power ball-sq-mtx-pow2*

by (*auto simp: plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def mat-def scaleR-vec-def axis-def plus-vec-def*)

lemma *bouncing-ball-sq-mtx*:

$\lceil \lambda s. 0 \leq s\$1 \wedge s\$1 = h \wedge s\$2 = 0 \wedge 0 > s\$3 \rceil \leq wp$

(*LOOP*

$((x' = (*_V)A \ \& \ (\lambda s. s\$1 \geq 0));$

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

$\text{INV } (\lambda s. 0 \leq s\$1 \wedge 0 > s\$3 \wedge 2 \cdot s\$3 \cdot s\$1 = 2 \cdot s\$3 \cdot h + (s\$2 \cdot s\$2)))$

$\lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$

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

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

apply(*simp add: sq-mtx-vec-mult-eq*)

unfolding *UNIV-3* **apply**(*simp add: exp-ball-sq-mtx-simps, safe*)

using *bb-real-arith(2)* **apply**(*force simp: add commute mult commute*)

using *bb-real-arith(3)* **by** (*force simp: add commute mult commute*)

no-notation *fball* (*f*)

and *ball-flow* (φ)

and *ball-sq-mtx* (A)

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 . if $i = 2$ then 1 else (if $i = 1$ then $-a * (s\$1 - L)$ else 0))

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

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

— Verified with the flow.

lemma *norm-diff-temp-dyn*: $0 < a \implies \|f$ 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 $L)$ *UNIV UNIV* (φ a L)

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

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

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

proof–

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

qed

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

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

proof–

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

qed

lemmas *fbox-temp-dyn* = *local-flow.fbox-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
 $((\mathcal{Q} ::= (\lambda s. 0)); (\mathcal{J} ::= (\lambda s. s\$1)));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq T_{min} + 1) THEN (\mathcal{A} ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq T_{max} - 1) THEN (\mathcal{A} ::= (\lambda s. 0)) ELSE skip));$
 — dynamics
 $(IF (\lambda s. s\$4 = 0) THEN (x' = (f a 0) \ \& \ (\lambda s. s\$2 \leq - (\ln (T_{min}/s\$3))/a)$
on $\{0..t\}$ *UNIV @ 0*)
 $ELSE (x' = (f a L) \ \& \ (\lambda s. s\$2 \leq - (\ln ((L - T_{max})/(L - s\$3))/a)$ *on* $\{0..t\}$
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 T_{min}*]
 and *temp-dyn-down-real-arith*[*OF assms*(1,3), *of - T_{max}*] **by auto**

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

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 *kat2rel*

imports

../hs-prelims-ka

../hs-prelims-dyn-sys

begin

0.9.1 Store and Hoare triples

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

— 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* (τ)
and *proto-near-quantale-class.bres* (**infixr** $\rightarrow 60$)

notation *Id* (*skip*)

— Canonical lifting from predicates to relations and its simplification rules

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*

— Meaning of the relational hoare triple

lemma *rel-kat-H*: *rel-kat.Hoare* $\lceil P \rceil\ X\ \lceil Q \rceil \longleftrightarrow (\forall s\ s'. P\ s \longrightarrow (s, s') \in X \longrightarrow Q\ s')$
by (*simp add: rel-kat.Hoare-def, auto simp add: p2r-def*)

— Hoare triple for skip and a simp-rule

lemma *H-skip*: *rel-kat.Hoare* $\lceil P \rceil\ skip\ \lceil P \rceil$
using *rel-kat.H-skip* **by** *blast*

lemma *sH-skip*[*simp*]: *rel-kat.Hoare* $\lceil P \rceil\ skip\ \lceil Q \rceil \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
unfolding *rel-kat-H* **by** *simp*

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

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

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

lemma *H-assign*: $P = (\lambda s. Q\ (\chi\ j. (((\$)\ s)(x := (e\ s)))\ j)) \implies \text{rel-kat.Hoare } \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil$
unfolding *rel-kat-H assign-def vec-upd-def* **by** *force*

lemma $sH\text{-assign}[simp]$: $rel\text{-kat.Hoare } [P] (x ::= e) [Q] \longleftrightarrow (\forall s. P\ s \longrightarrow Q\ (\chi\ j. (((\$)\ s)(x := (e\ s))))\ j))$

unfolding $rel\text{-kat-H}\ vec\text{-upd-def}\ assign\text{-def}$ **by** $(auto\ simp: fun\text{-upd-def})$

— Next, the Hoare rule of the composition

lemma $H\text{-seq}$: $rel\text{-kat.Hoare } [P] X [R] \Longrightarrow rel\text{-kat.Hoare } [R] Y [Q] \Longrightarrow rel\text{-kat.Hoare } [P] (X ; Y) [Q]$

by $(auto\ intro: rel\text{-kat.H-seq})$

lemma $sH\text{-seq}$:

$rel\text{-kat.Hoare } [P] (X ; Y) [Q] = rel\text{-kat.Hoare } [P] (X) [\lambda s. \forall s'. (s, s') \in Y \longrightarrow Q\ s']$

unfolding $rel\text{-kat-H}$ **by** $auto$

— Rewriting the Hoare rule for the conditional statement

abbreviation $cond\text{-sugar} :: 'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ ($IF - THEN - ELSE - [64, 64] 63$)

where $IF\ B\ THEN\ X\ ELSE\ Y \equiv rel\text{-kat.kat-cond } [B] X Y$

lemma $H\text{-cond}$: $rel\text{-kat.Hoare } [P \sqcap B] X [Q] \Longrightarrow rel\text{-kat.Hoare } [P \sqcap \neg B] Y [Q] \Longrightarrow$

$rel\text{-kat.Hoare } [P] (IF\ B\ THEN\ X\ ELSE\ Y) [Q]$

by $(rule\ rel\text{-kat.H-cond}, auto\ simp: rel\text{-kat-H})$

lemma $sH\text{-cond}[simp]$: $rel\text{-kat.Hoare } [P] (IF\ B\ THEN\ X\ ELSE\ Y) [Q] \longleftrightarrow$

$(rel\text{-kat.Hoare } [P \sqcap B] X [Q] \wedge rel\text{-kat.Hoare } [P \sqcap \neg B] Y [Q])$

by $(auto\ simp: rel\text{-kat.H-cond-iff}\ rel\text{-kat-H})$

— Rewriting the Hoare rule for the while loop

abbreviation $while\text{-inv-sugar} :: 'a\ pred \Rightarrow 'a\ pred \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ ($WHILE - INV - DO - [64, 64, 64] 63$)

where $WHILE\ B\ INV\ I\ DO\ X \equiv rel\text{-kat.kat-while-inv } [B] [I] X$

lemma $sH\text{-while-inv}$: $\forall s. P\ s \longrightarrow I\ s \Longrightarrow \forall s. I\ s \wedge \neg B\ s \longrightarrow Q\ s \Longrightarrow rel\text{-kat.Hoare } [I \sqcap B] X [I]$

$\Longrightarrow rel\text{-kat.Hoare } [P] (WHILE\ B\ INV\ I\ DO\ X) [Q]$

by $(rule\ rel\text{-kat.H-while-inv}, auto\ simp: p2r\text{-def}\ rel\text{-kat.Hoare-def}, fastforce)$

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

abbreviation $loopi\text{-sugar} :: 'a\ rel \Rightarrow 'a\ pred \Rightarrow 'a\ rel$ ($LOOP - INV - [64, 64] 63$)

where $LOOP\ X\ INV\ I \equiv rel\text{-kat.kat-loop-inv } X [I]$

lemma $H\text{-loop}$: $rel\text{-kat.Hoare } [P] X [P] \Longrightarrow rel\text{-kat.Hoare } [P] (LOOP\ X\ INV\ I) [P]$

by (*auto intro: rel-kat.H-loop*)

lemma *H-loopI*: *rel-kat.Hoare* $\lceil I \rceil$ *X* $\lceil I \rceil \Rightarrow \lceil P \rceil \subseteq \lceil I \rceil \Rightarrow \lceil I \rceil \subseteq \lceil Q \rceil \Rightarrow$
rel-kat.Hoare $\lceil P \rceil$ (*LOOP X INV I*) $\lceil Q \rceil$
using *rel-kat.H-loop-inv* [*of* $\lceil P \rceil$ $\lceil I \rceil$ *X* $\lceil Q \rceil$] **by** *auto*

0.9.2 Verification of hybrid programs

— Verification by providing evolution

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

lemma *H-g-evol*:

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
assumes $P = (\lambda s. (\forall t \in T. (\forall \tau \in \text{down } T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$
shows *rel-kat.Hoare* $\lceil P \rceil$ (*EVOL* φ *G T*) $\lceil Q \rceil$
unfolding *rel-kat-H g-evol-def g-orbit-eq* **using** *assms* **by** *clarsimp*

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

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows *rel-kat.Hoare* $\lceil P \rceil$ (*EVOL* φ *G T*) $\lceil Q \rceil = (\forall s. \ P \ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$
unfolding *rel-kat-H g-evol-def g-orbit-eq* **by** *auto*

— Verification by providing solutions

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

lemma *H-g-orbital*:

$P = (\lambda s. (\forall X \in \text{ivp-sols} \ (\lambda t. \ f) \ T \ S \ t_0 \ s. \ \forall t \in T. (\forall \tau \in \text{down } T \ t. \ G \ (X \ \tau)) \longrightarrow Q \ (X \ t))) \Rightarrow$
rel-kat.Hoare $\lceil P \rceil$ $(x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0)$ $\lceil Q \rceil$
unfolding *rel-kat-H g-ode-def g-orbital-eq* **by** *clarsimp*

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

context *local-flow*

begin

lemma *H-g-ode*:

assumes $P = (\lambda s. \ s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s)))$
shows *rel-kat.Hoare* $\lceil P \rceil$ $(x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ 0)$ $\lceil Q \rceil$

proof(*unfold rel-kat-H g-ode-def g-orbital-eq assms, clarsimp*)
fix $s\ t\ X$
assume $\text{hyps}: t \in T \ \forall x. x \in T \wedge x \leq t \longrightarrow G\ (X\ x)\ X \in \text{Sols}\ (\lambda t. f)\ T\ S\ 0\ s$
and $\text{main}: s \in S \longrightarrow (\forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
have $s \in S$
using *ivp-solsD[OF hyps(3)] init-time by auto*
hence $\forall \tau \in \text{down}\ T\ t. X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps by blast*
thus $Q\ (X\ t)$
using $\text{main}\ \langle s \in S \rangle \text{ hyps by fastforce}$
qed

lemma *sH-g-ode: rel-kat.Hoare* $\lceil P \rceil\ (x' = f \ \&\ G \text{ on } T\ S\ @\ 0)\ \lceil Q \rceil =$
 $(\forall s \in S. P\ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down}\ T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$

proof(*unfold sH-g-orbital, clarsimp, safe*)
fix $s\ t$
assume $\text{hyps}: s \in S\ P\ s\ t \in T\ \forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)$
and $\text{main}: \forall s. P\ s \longrightarrow (\forall X \in \text{Sols}\ (\lambda t. f)\ T\ S\ 0\ s. \forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
hence $(\lambda t. \varphi\ t\ s) \in \text{Sols}\ (\lambda t. f)\ T\ S\ 0\ s$
using *in-ivp-sols by blast*
thus $Q\ (\varphi\ t\ s)$
using $\text{main}\ \text{hyps by fastforce}$
next
fix $s\ X\ t$
assume $\text{hyps}: P\ s\ X \in \text{Sols}\ (\lambda t. f)\ T\ S\ 0\ s\ t \in T\ \forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
and $\text{main}: \forall s \in S. P\ s \longrightarrow (\forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
hence $\text{obs}: s \in S$
using *ivp-sols-def[of $\lambda t. f$] init-time by auto*
hence $\forall \tau \in \text{down}\ T\ t. X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps by blast*
thus $Q\ (X\ t)$
using $\text{hyps}\ \text{main}\ \text{obs by auto}$
qed

lemma *sH-g-ode-ivl: $\tau \geq 0 \implies \tau \in T \implies \text{rel-kat.Hoare } \lceil P \rceil\ (x' = f \ \&\ G \text{ on } \{0..\tau\}\ S\ @\ 0)\ \lceil Q \rceil =$*

$(\forall s \in S. P\ s \longrightarrow (\forall t \in \{0..\tau\}. (\forall \tau \in \{0..t\}. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)))$
proof(*unfold sH-g-orbital, clarsimp, safe*)
fix $s\ t$
assume $\text{hyps}: 0 \leq \tau\ \tau \in T\ s \in S\ P\ s\ t \in \{0..\tau\}\ \forall \tau \in \{0..t\}. G\ (\varphi\ \tau\ s)$
and $\text{main}: \forall s. P\ s \longrightarrow (\forall X \in \text{Sols}\ (\lambda t. f)\ \{0..\tau\}\ S\ 0\ s. \forall t \in \{0..\tau\}. (\forall \tau'. 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow G\ (X\ \tau')) \longrightarrow Q\ (X\ t))$
hence $(\lambda t. \varphi\ t\ s) \in \text{Sols}\ (\lambda t. f)\ \{0..\tau\}\ S\ 0\ s$
using *in-ivp-sols-ivl closed-segment-eq-real-ivl[of 0 τ] by force*
thus $Q\ (\varphi\ t\ s)$

```

    using main hyps by fastforce
next
  fix s X t
  assume hyps:  $0 \leq \tau \wedge \tau \in T \wedge P \ s \wedge X \in \text{Sols} \ (\lambda t. f) \ \{0.. \tau\} \wedge S \ 0 \ s \ t \in \{0.. \tau\}$ 
     $\forall \tau'. \ 0 \leq \tau' \wedge \tau' \leq \tau \wedge \tau' \leq t \longrightarrow G \ (X \ \tau')$ 
  and main:  $\forall s \in S. \ P \ s \longrightarrow (\forall t \in \{0.. \tau\}. \ (\forall \tau \in \{0.. t\}. \ G \ (\varphi \ \tau \ s)) \longrightarrow Q \ (\varphi \ t \ s))$ 
  hence  $s \in S$ 
  using ivp-sols-def[of  $\lambda t. f$ ] init-time by auto
  have obs1:  $\forall \tau \in \text{down} \ \{0.. \tau\} \ t. \ D \ X = (\lambda t. f \ (X \ t)) \text{ on } \{0 \dashv \tau\}$ 
  apply (clarsimp, rule has-vderiv-on-subset)
  using ivp-solsD(1)[OF hyps(4)] by (auto simp: closed-segment-eq-real-ivl)
  have obs2:  $X \ 0 = s \wedge \forall \tau \in \text{down} \ \{0.. \tau\} \ t. \ X \in \{0 \dashv \tau\} \rightarrow S$ 
  using ivp-solsD(2,3)[OF hyps(4)] by (auto simp: closed-segment-eq-real-ivl)
  have  $\forall \tau \in \text{down} \ \{0.. \tau\} \ t. \ \tau \in T$ 
  using subintervalI[OF init-time  $\langle \tau \in T \rangle$ ] by (auto simp: closed-segment-eq-real-ivl)
  hence  $\forall \tau \in \text{down} \ \{0.. \tau\} \ t. \ X \ \tau = \varphi \ \tau \ s$ 
  using obs1 obs2 apply (clarsimp)
  by (rule eq-solution-ivl) (auto simp: closed-segment-eq-real-ivl)
  thus  $Q \ (X \ t)$ 
  using hyps main  $\langle s \in S \rangle$  by auto
qed

```

lemma *sH-orbit*:

```

  rel-kat.Hoare  $\lceil P \rceil \ ((s, s') \mid s \ s'. \ s' \in \gamma^\varphi \ s) \ \lceil Q \rceil = (\forall s \in S. \ P \ s \longrightarrow (\forall t \in T. \ Q \ (\varphi \ t \ s)))$ 
  using sH-g-ode unfolding orbit-def g-ode-def by auto

```

end

— Verification with differential invariants

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

lemma *sH-g-orbital-guard*:

```

  assumes  $R = (\lambda s. \ G \ s \wedge Q \ s)$ 
  shows rel-kat.Hoare  $\lceil P \rceil \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \lceil Q \rceil = \text{rel-kat.Hoare} \ \lceil P \rceil \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \lceil R \rceil$ 
  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 rel-kat.Hoare  $\lceil I \rceil \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \lceil I \rceil$  and  $\lceil I \rceil \leq \lceil Q \rceil$ 
  shows rel-kat.Hoare  $\lceil P \rceil \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ \lceil Q \rceil$ 
  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]*: *rel-kat.Hoare* $\lceil I \rceil$ $(x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0)$ $\lceil I \rceil =$
diff-invariant $I \ f \ T \ S \ t_0 \ G$

unfolding *diff-invariant-eq rel-kat-H g-orbital-eq g-ode-def* **by** *auto*

lemma *H-g-ode-inv*: *rel-kat.Hoare* $\lceil I \rceil$ $(x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0)$ $\lceil I \rceil \implies \lceil P \rceil \leq$
 $\lceil I \rceil \implies$

$\lceil \lambda s. I \ s \ \wedge \ G \ s \rceil \leq \lceil Q \rceil \implies \text{rel-kat.Hoare } \lceil P \rceil \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0 \ DINV$
 $I) \ \lceil Q \rceil$

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*)

0.9.3 Refinement Components

— Skip

lemma *R-skip*: $(\forall s. P \ s \longrightarrow Q \ s) \implies Id \leq \text{rel-R } \lceil P \rceil \ \lceil Q \rceil$
by (*simp add: rel-rkat.R2 rel-kat-H*)

— 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*

— 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*: $(\forall s. P \ s \longrightarrow Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j)) \implies (x ::=$
 $e) \leq \text{rel-R } \lceil P \rceil \ \lceil Q \rceil$

unfolding *sH-assign[symmetric]* **by** (*rule rel-rkat.R2*)

lemma *R-assignl*: $P = (\lambda s. R \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)) \implies (x ::= e) ; \text{rel-R}$
 $\lceil R \rceil \ \lceil Q \rceil \leq \text{rel-R } \lceil P \rceil \ \lceil Q \rceil$

apply(*rule-tac* $R = R$ **in** *R-seq-rule*)

by (*rule-tac* *R-assign-rule*, *simp-all*)

lemma *R-assignr*: $R = (\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)) \implies \text{rel-R } \lceil P \rceil \ \lceil R \rceil ; (x$
 $::= e) \leq \text{rel-R } \lceil P \rceil \ \lceil Q \rceil$

apply(*rule-tac* $R = R$ **in** *R-seq-rule*, *simp*)

by (*rule-tac* *R-assign-rule*, *simp*)

lemma $(x ::= e) ; \text{rel-R } \lceil Q \rceil \lceil Q \rceil \leq \text{rel-R } \lceil (\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)) \rceil \lceil Q \rceil$
by (*rule R-assignl*) *simp*

lemma $\text{rel-R } \lceil Q \rceil \lceil (\lambda s. Q (\chi j. (((\$) s)(x := e s)) j)) \rceil ; (x ::= e) \leq \text{rel-R } \lceil Q \rceil \lceil Q \rceil$
by (*rule R-assignr*) *simp*

— Conditional

lemma *R-cond*: $(\text{IF } B \text{ THEN } \text{rel-R } \lceil \lambda s. B s \wedge P s \rceil \lceil Q \rceil \text{ ELSE } \text{rel-R } \lceil \lambda s. \neg B s \wedge P s \rceil \lceil Q \rceil) \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
using *rel-rkat.R-cond*[*of* $\lceil B \rceil \lceil P \rceil \lceil Q \rceil$] **by** *simp*

lemma *R-cond-mono*: $X \leq X' \implies Y \leq Y' \implies (\text{IF } P \text{ THEN } X \text{ ELSE } Y) \leq \text{IF } P \text{ THEN } X' \text{ ELSE } Y'$
by (*auto simp: rel-kat.kat-cond-def*)

— While loop

lemma *R-while*: $\text{WHILE } Q \text{ INV } I \text{ DO } (\text{rel-R } \lceil \lambda s. P s \wedge Q s \rceil \lceil P \rceil) \leq \text{rel-R } \lceil P \rceil \lceil \lambda s. P s \wedge \neg Q s \rceil$
unfolding *rel-kat.kat-while-inv-def* **using** *rel-rkat.R-while*[*of* $\lceil Q \rceil \lceil P \rceil$] **by** *simp*

lemma *R-while-mono*: $X \leq X' \implies (\text{WHILE } P \text{ INV } I \text{ DO } X) \subseteq \text{WHILE } P \text{ INV } I \text{ DO } X'$
by (*simp add: rel-kat.kat-while-inv-def rel-kat.kat-while-def rel-uq.mult-isol rel-uq.mult-isor rel-ka.star-iso*)

— Finite loop

lemma *R-loop*: $X \leq \text{rel-R } \lceil I \rceil \lceil I \rceil \implies \lceil P \rceil \leq \lceil I \rceil \implies \lceil I \rceil \leq \lceil Q \rceil \implies \text{LOOP } X \text{ INV } I \leq \text{rel-R } \lceil P \rceil \lceil Q \rceil$
unfolding *rel-rkat.spec-def* **using** *H-loopI* **by** *blast*

lemma *R-loop-mono*: $X \leq X' \implies \text{LOOP } X \text{ INV } I \subseteq \text{LOOP } X' \text{ INV } I$
unfolding *rel-kat.kat-loop-inv-def* **by** (*simp add: rel-ka.star-iso*)

— Evolution command (flow)

lemma *R-g-evol*:
fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $(\text{EVOL } \varphi \text{ } G \text{ } T) \leq \text{rel-R } \lceil \lambda s. \forall t \in T. (\forall \tau \in \text{down } T \text{ } t. G (\varphi \tau s)) \longrightarrow P (\varphi t s) \rceil \lceil P \rceil$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-evol, simp*)

lemma *R-g-evol-rule*:
fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$

shows $(\forall s. P\ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \implies$
 $(\text{EVOL } \varphi\ G\ T) \leq \text{rel-R } [P] \ [Q]$
unfolding *sH-g-evol[symmetric] rel-rkat.spec-def* .

lemma *R-g-evoll*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \implies$
 $(\text{EVOL } \varphi\ G\ T) ; \text{rel-R } [R] \ [Q] \leq \text{rel-R } [P] \ [Q]$
apply(*rule-tac R=R in R-seq-rule*)
by (*rule-tac R-g-evol-rule, simp-all*)

lemma *R-g-evolr*:

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)) \implies$
 $\text{rel-R } [P] \ [R] ; (\text{EVOL } \varphi\ G\ T) \leq \text{rel-R } [P] \ [Q]$
apply(*rule-tac R=R in R-seq-rule, simp*)
by (*rule-tac R-g-evol-rule, simp*)

lemma

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{EVOL } \varphi\ G\ T ; \text{rel-R } [Q] \ [Q] \leq \text{rel-R } [\lambda s. \forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)] \ [Q]$
by (*rule R-g-evoll simp*)

lemma

fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{rel-R } [Q] \ [\lambda s. \forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s)] ; \text{EVOL } \varphi\ G\ T \leq \text{rel-R } [Q] \ [Q]$
by (*rule R-g-evolr simp*)

— Evolution command (ode)

context *local-flow*

begin

lemma *R-g-ode*: $(x' = f \ \& \ G \text{ on } T\ S \ @ \ 0) \leq \text{rel-R } [\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s))] \ [P]$
unfolding *rel-rkat.spec-def* **by** (*rule H-g-ode, simp*)

lemma *R-g-ode-rule*: $(\forall s \in S. P\ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \implies$
 $(x' = f \ \& \ G \text{ on } T\ S \ @ \ 0) \leq \text{rel-R } [P] \ [Q]$
unfolding *sH-g-ode[symmetric]* **by** (*rule rel-rkat.R2*)

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow R\ (\varphi\ t\ s)) \implies$
 $(x' = f \ \& \ G \text{ on } T\ S \ @ \ 0) ; \text{rel-R } [R] \ [Q] \leq \text{rel-R } [P] \ [Q]$
apply(*rule-tac R=R in R-seq-rule*)
by (*rule-tac R-g-ode-rule, simp-all*)

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s)) \Longrightarrow$

$\text{rel-}R \llbracket P \rrbracket \llbracket R \rrbracket; (x' = f \ \& \ G \text{ on } T S @ 0) \leq \text{rel-}R \llbracket P \rrbracket \llbracket Q \rrbracket$
apply(*rule-tac* $R=R$ **in** *R-seq-rule*, *simp*)
by (*rule-tac* *R-g-ode-rule*, *simp*)

lemma $(x' = f \ \& \ G \text{ on } T S @ 0); \text{rel-}R \llbracket Q \rrbracket \llbracket Q \rrbracket \leq \text{rel-}R \llbracket \lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s) \rrbracket \llbracket Q \rrbracket$
by (*rule* *R-g-odel*) *simp*

lemma $\text{rel-}R \llbracket Q \rrbracket \llbracket \lambda s. \forall t \in T. (\forall \tau \in \text{down } T t. G (\varphi \tau s)) \longrightarrow Q (\varphi t s) \rrbracket; (x' = f \ \& \ G \text{ on } T S @ 0) \leq \text{rel-}R \llbracket Q \rrbracket \llbracket Q \rrbracket$
by (*rule* *R-g-oder*) *simp*

lemma *R-g-ode-ivl*:

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

end

— Evolution command (invariants)

lemma *R-g-ode-inv*: *diff-invariant* $I f T S t_0 G \Longrightarrow \llbracket P \rrbracket \leq \llbracket I \rrbracket \Longrightarrow \llbracket \lambda s. I s \wedge G s \rrbracket \leq \llbracket Q \rrbracket \Longrightarrow$
 $(x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I) \leq \text{rel-}R \llbracket P \rrbracket \llbracket Q \rrbracket$
unfolding *rel-rkat.spec-def* **by** (*auto simp: H-g-ode-inv*)

0.9.4 Derivation of the rules of dL

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

lemma *diff-solve-axiom*:

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

lemma *diff-solve-rule*:

assumes *local-flow* $f T \text{ UNIV } \varphi$
and $\forall s. P s \longrightarrow (\forall t \in T. (\mathcal{P} (\lambda t. \varphi t s) (\text{down } T t) \subseteq \{s. G s\}) \longrightarrow Q (\varphi t s))$
shows *rel-kat.Hoare* $\llbracket P \rrbracket (x' = f \ \& \ G \text{ on } T \text{ UNIV } @ 0) \llbracket Q \rrbracket$
using *assms* **by**(*subst local-flow.sH-g-ode, auto*)

```

lemma diff-weak-rule:
  assumes  $\lceil G \rceil \leq \lceil Q \rceil$ 
  shows rel-kat.Hoare  $\lceil P \rceil (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \lceil Q \rceil$ 
  using assms unfolding g-orbital-eq rel-kat-H ivp-sols-def g-ode-def by auto

```

```

lemma diff-cut-rule:
  assumes Thyp: is-interval  $T \ t_0 \in T$ 
  and wp-C: rel-kat.Hoare  $\lceil P \rceil (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \lceil C \rceil$ 
  and wp-Q: rel-kat.Hoare  $\lceil P \rceil (x' = f \ \& \ (\lambda s. G \ s \wedge C \ s) \text{ on } T \ S \ @ \ t_0) \lceil Q \rceil$ 
  shows rel-kat.Hoare  $\lceil P \rceil (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \lceil Q \rceil$ 
proof(subst rel-kat-H, 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  $t \in T$ 
  and  $x-ivp$ :  $X \in ivp-sols \ (\lambda t. f) \ T \ S \ t_0 \ s$ 
  and guard-x:  $\forall x. x \in T \wedge x \leq t \longrightarrow G \ (X \ x)$ 
  have  $\forall t \in (down \ T \ t). X \ t \in g-orbital \ f \ G \ T \ S \ t_0 \ s$ 
  using g-orbitalI[OF x-ivp] guard-x by auto
  hence  $\forall t \in (down \ T \ t). C \ (X \ t)$ 
  using wp-C  $\langle P \ s \rangle$  by (subst (asm) rel-kat-H, auto simp: g-ode-def)
  hence  $X \ t \in g-orbital \ f \ (\lambda s. G \ s \wedge C \ s) \ T \ S \ t_0 \ s$ 
  using guard-x  $\langle t \in T \rangle$  by (auto intro!: g-orbitalI x-ivp)
  thus  $Q \ (X \ t)$ 
  using  $\langle P \ s \rangle$  wp-Q by (subst (asm) rel-kat-H) (auto simp: g-ode-def)
qed

```

```

abbreviation g-global-ode ::  $((a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel} \ ((1x' = - \ \& \ -))$ 
  where  $(x' = f \ \& \ G) \equiv (x' = f \ \& \ G \text{ on } UNIV \ UNIV \ @ \ 0)$ 

```

```

abbreviation g-global-ode-inv ::  $((a::banach) \Rightarrow 'a) \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ pred} \Rightarrow 'a \text{ rel}$ 
   $((1x' = - \ \& \ - \ DINV \ -))$  where  $(x' = f \ \& \ G \ DINV \ I) \equiv (x' = f \ \& \ G \text{ on } UNIV$ 
   $UNIV \ @ \ 0 \ DINV \ I)$ 

```

end

0.9.5 Examples

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

```

theory kat2rel-examples
  imports kat2rel

```

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 $pendulum-dyn: rel-kat.Hoare\ [\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]\ (EVOL\ \varphi\ G\ T)$
 $[\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]$
by $simp$

— Verified with differential invariants

lemma $pendulum-inv: rel-kat.Hoare$
 $[\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]\ (x' = f \ \&\ G)\ [\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]$
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 $pendulum-flow: rel-kat.Hoare$
 $[\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]\ (x' = f \ \&\ G)\ [\lambda s.\ r^2 = (s\$1)^2 + (s\$2)^2]$
by $(simp\ only:\ local-flow.sH-g-ode[OF\ local-flow-pend],\ simp)$

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

lemma *fball-invariant*:
fixes $g \ h :: real$
defines *dinv*: $I \equiv (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - (s\$2 \cdot s\$2) = 0)$
shows *diff-invariant I (f g) UNIV UNIV 0 G*
unfolding *dinv* **apply**(*rule diff-invariant-rules, simp, simp, clarify*)
by(*auto intro!: poly-derivatives*)

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies rel\text{-kat.Hoare}$
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil$
 $(LOOP$
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0) \ DINV \ (\lambda s. 2 \cdot g \cdot s\$1 - 2 \cdot g \cdot h - s\$2 \cdot s\$2$
 $= 0));$
 $(IF \ (\lambda s. s\$1 = 0) \ THEN \ (2 ::= (\lambda s. - s\$2)) \ ELSE \ skip))$
 $INV \ (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \lceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
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*: $g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$

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* = $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
apply(*subst* *Rat.sign-simps*(18))+
by(*auto* *simp: semiring-normalization-rules*(29))
also **have** ... = $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$ (*is* ... = ?*middle*)
by(*subst invar, simp*)
finally **have** ?*lhs* = ?*middle*.
moreover
{**have** ?*rhs* = $g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$
by (*simp* *add: Groups.mult-ac*(2,3) *semiring-class.distrib-left*)
also **have** ... = ?*middle*
by (*simp* *add: semiring-normalization-rules*(29))
finally **have** ?*rhs* = ?*middle*.**}**
ultimately **show** ?*thesis* **by** *auto*
qed

lemma *bouncing-ball-dyn*: $g < 0 \implies h \geq 0 \implies \text{rel-kat.Hoare}$
 $\lceil \lambda s. s\$1 = h \wedge s\$2 = 0 \rceil$
(*LOOP*
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $\rceil \lambda s. 0 \leq s\$1 \wedge s\$1 \leq h \rceil$
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* (φ *g*)
apply(*unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def vec-eq-iff, clarsimp*)
apply(*rule-tac x=1/2 in exI, clarsimp, rule-tac x=1 in exI*)
apply(*simp add: dist-norm norm-vec-def L2-set-def UNIV-2*)
by (*auto simp: forall-2 intro!: poly-derivatives*)

lemma *bouncing-ball-flow*: $g < 0 \implies h \geq 0 \implies \text{rel-kat.Hoare}$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0]$
 $(\text{LOOP}$
 $((x' = f\ g \ \& \ (\lambda s. s\$1 \geq 0));$
 $(\text{IF } (\lambda s. s\$1 = 0) \text{ THEN } (2 ::= (\lambda s. - s\$2)) \text{ ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
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(*subst local-flow.sH-g-ode[OF local-flow-ball]*)
apply(*force simp: bb-real-arith*)
by (*rule H-cond*) (*auto simp: bb-real-arith*)

— Refined with annotated dynamics

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

lemma *R-bouncing-ball-dyn*:
assumes $g < 0$ **and** $h \geq 0$
shows *rel-R* $[\lambda s. s\$1 = h \wedge s\$2 = 0] \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h] \geq$
 $(\text{LOOP}$
 $((\text{EVOL } (\varphi\ g) (\lambda s. s\$1 \geq 0) \ T);$
 $(\text{IF } (\lambda s. s\$1 = 0) \text{ THEN } (2 ::= (\lambda s. - s\$2)) \text{ ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2))$
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* (φ)

Thermostat

A thermostat has a chronometer, a thermometer and a switch to turn on and off a heater. At most every τ minutes, it sets its chronometer to 0, it registers the room temperature, and it turns the heater on (or off) based on this reading. The temperature follows the ODE $T' = -a * (T - U)$ where $U = L \geq 0$ when the heater is on, and $U = 0$ when it is off. We use 1 to denote the room's temperature, 2 is time as measured by the thermostat's chronometer, and 3 is a variable to save temperature measurements. Finally, 4 states whether the heater is on ($s4 = 1$) or off ($s4 = 0$). We prove that the thermostat keeps the room's temperature between $Tmin$ and $Tmax$.

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

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

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

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

— Verified with the flow

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

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

assume *a1*: $0 < a$

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

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

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

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

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

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

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

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

qed

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

lemma *thermostat-flow*:

```

  assumes  $0 < a$  and  $0 \leq \tau$  and  $0 < Tmin$  and  $Tmax < L$ 
  shows rel-kat.Hoare  $[I \text{ } Tmin \text{ } Tmax]$ 
    (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
        ( $x' = f \ a \ 0 \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on} \ \{0..\tau\} \ UNIV \ @ \ 0$ )
      ELSE
        ( $x' = f \ a \ L \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on} \ \{0..\tau\} \ UNIV \ @ \ 0$ ))
    ) INV  $I \text{ } Tmin \ Tmax$ )
   $[I \text{ } Tmin \ Tmax]$ 
  apply(rule H-loopI)
  apply(rule-tac R= $\lambda s. I \text{ } Tmin \ Tmax \ s \wedge s\$2=0 \wedge s\$3 = s\$1$  in H-seq)
  apply(rule-tac R= $\lambda s. I \text{ } Tmin \ Tmax \ s \wedge s\$2=0 \wedge s\$3 = s\$1$  in H-seq)
  apply(rule-tac R= $\lambda s. I \text{ } Tmin \ Tmax \ s \wedge s\$2=0$  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-R  $[\lambda s. s\$4 = 0 \wedge I \text{ } Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \text{ } Tmin \ Tmax] \geq$ 

```

```

  (x' = f a 0 & G Tmin Tmax a 0 on {0..τ} 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 ≤ τ and 0 < Tmin and Tmax < L
  shows rel-R [λs. s$4 ≠ 0 ∧ I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1] [I Tmin
Tmax] ≥
  (x' = f a L & G Tmin Tmax a L on {0..τ} 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 ≤ τ and 0 < Tmin and Tmax < L
  shows rel-R [λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1] [I Tmin Tmax] ≥
  (IF (λs. s$4 = 0) THEN
    (x' = f a 0 & G Tmin Tmax a 0 on {0..τ} UNIV @ 0)
  ELSE
    (x' = f a L & G Tmin Tmax a L on {0..τ} 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*: $rel-R [I Tmin Tmax] [λs. I Tmin Tmax s ∧ s$2 = 0] ≥ (2 ::= (λs. 0))$
by (auto simp: R-assign-rule)

lemma *R-therm-assign2*:

```

  rel-R [λs. I Tmin Tmax s ∧ s$2 = 0] [λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3
= s$1] ≥ (3 ::= (λs. s$1))
  by (auto simp: R-assign-rule)

```

lemma *R-therm-ctrl*:

```

  rel-R [I Tmin Tmax] [λs. I Tmin Tmax s ∧ s$2 = 0 ∧ s$3 = s$1] ≥
  (2 ::= (λs. 0));
  (3 ::= (λs. s$1));
  (IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
    (4 ::= (λs. 1))
  ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax - 1) THEN
    (4 ::= (λs. 0))
  ELSE skip)
  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*
 $\text{rel-}R \ [I \ Tmin \ Tmax] \ [\lambda s. \ I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1];$
 $\text{rel-}R \ [\lambda s. \ I \ Tmin \ Tmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ Tmin \ Tmax]$
 $\text{INV } I \ Tmin \ Tmax$)
by (*intro R-loop R-seq, simp-all*)

lemma *R-thermostat-flow*:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < Tmin$ **and** $Tmax < L$
shows $\text{rel-}R \ [I \ Tmin \ Tmax] \ [I \ Tmin \ Tmax] \geq$
 (*LOOP* (
 — control
 $(2 ::= (\lambda s. \ 0)); (3 ::= (\lambda s. \ s\$1));$
 $(\text{IF } (\lambda s. \ s\$4 = 0 \wedge s\$3 \leq Tmin + 1) \ \text{THEN}$
 $(4 ::= (\lambda s. \ 1))$
 $\text{ELSE IF } (\lambda s. \ s\$4 = 1 \wedge s\$3 \geq Tmax - 1) \ \text{THEN}$
 $(4 ::= (\lambda s. \ 0))$
 $\text{ELSE skip};$
 — dynamics
 $(\text{IF } (\lambda s. \ s\$4 = 0) \ \text{THEN}$
 $(x' = f \ a \ 0 \ \& \ G \ Tmin \ Tmax \ a \ 0 \ \text{on } \{0..\tau\} \ \text{UNIV } @ \ 0)$
 ELSE
 $(x' = f \ a \ L \ \& \ G \ Tmin \ Tmax \ a \ L \ \text{on } \{0..\tau\} \ \text{UNIV } @ \ 0))$
 $\text{INV } I \ Tmin \ Tmax$)
by (*intro order-trans[OF - R-therm-loop] R-loop-mono*
 $R\text{-seq-mono } R\text{-therm-ctrl } R\text{-therm-dyn}[OF \ \text{assms}]$)

no-notation *therm-vec-field* (f)
and *therm-flow* (φ)
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* :: $real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$ (*G*)
where $G \ Hm \ k \ s \equiv s\$2 \leq (Hm - s\$3)/k$

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

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

— Verified with the flow

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

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

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

lemma *tank-flow*:
assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *rel-kat.Hoare* [*I hmin hmax*]
(LOOP
— control
 $((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 $(IF (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) THEN (4 ::= (\lambda s. 1)) ELSE$
 $(IF (\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1) THEN (4 ::= (\lambda s. 0)) ELSE skip));$
— dynamics
 $(IF (\lambda s. s\$4 = 0) THEN (x' = f (c_i - c_o) \ \& \ G \ hmax (c_i - c_o) \ on \ \{0.. \tau\} \ UNIV$
@ 0)
 $ELSE (x' = f (-c_o) \ \& \ G \ hmin (-c_o) \ on \ \{0.. \tau\} \ UNIV @ 0)))$
INV I hmin hmax] [*I hmin hmax*]
apply(*rule H-loopI*)
apply(*rule-tac* $R=\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0 \wedge s\$3 = s\1 **in** *H-seq*)
apply(*rule-tac* $R=\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0 \wedge s\$3 = s\1 **in** *H-seq*)
apply(*rule-tac* $R=\lambda s. I \ hmin \ hmax \ s \wedge s\$2=0$ **in** *H-seq, simp, simp*)

apply(rule *H-cond*, simp-all add: *H-g-ode-tank*[*OF* *assms*(1)])
using *assms tank-arith*[*OF* - *assms*(2,3)] **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:

$0 \leq \tau \implies \text{diff-invariant } (dI \text{ } hmin \text{ } hmax \text{ } k) \text{ } (f \text{ } k) \text{ } \{0.. \tau\} \text{ } UNIV \text{ } 0 \text{ } Guard$
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*)
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** $b: hmin \leq y_0$ **and** $g: \tau \leq (hmax - y_0) / (c_i - c_o)$
shows $hmin \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq hmax$
proof—
have $(c_i - c_o) \cdot \tau \leq (hmax - y_0)$
using *g* *assms*(2,3) **by** (*metis diff-gt-0-iff-gt mult.commute pos-le-divide-eq*)
thus $(c_i - c_o) \cdot \tau + y_0 \leq hmax$
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 linordered-field-class.sign-simps*(39) *mult-less-cancel-right*)

qed

lemma *tank-inv*:

assumes $0 \leq \tau$ **and** $0 < c_o$ **and** $c_o < c_i$
shows *rel-kat.Hoare* [*I* *hmin* *hmax*]
(*LOOP*
— *control*
 $((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1) \text{ THEN } (4 ::= (\lambda s. 1)) \text{ ELSE$

```

  (IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$ ) THEN ( $4 ::= (\lambda s. 0)$ ) ELSE skip));
  — dynamics
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = f(c_i - c_o) \ \& \ G \ hmax(c_i - c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \ hmin \ hmax \ (c_i - c_o)))$ )
  ELSE
    ( $x' = f(-c_o) \ \& \ G \ hmin(-c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \ hmin \ hmax \ (-c_o))$ ))
  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
      ( $x' = f(c_i - c_o) \ \& \ G \ hmax(c_i - c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \ hmin \ hmax \ (c_i - c_o))$ )
    ELSE
      ( $x' = f(-c_o) \ \& \ G \ hmin(-c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \ hmin \ hmax \ (-c_o))$ ))
    INV  $I \ hmin \ hmax$  (is LOOP (?ctrl;?dyn) INV -  $\leq$  ?ref)
  )

```

proof—

— First we refine the control.

```

  let ?Icctrl =  $\lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ 
  and ?cond =  $\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$ 
  have ifbranch1:  $4 ::= (\lambda s. 1) \leq \text{rel-R } \lceil \lambda s. ?cond \ s \wedge ?Icctrl \ s \rceil \lceil ?Icctrl \rceil$  (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-R  $\lceil \lambda s. \neg ?cond \ s \wedge ?Icctrl \ s \rceil \lceil ?Icctrl \rceil$  (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-R* [?*Ictrl*]
[?*Ictrl*] (**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-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-R* [*I hmin hmax*] [?*Ictrl*]
apply(rule-tac *R=?Ictrl in R-seq-rule*)
apply(rule-tac *R= $\lambda s. I hmin hmax s \wedge s\$2 = 0$ in R-seq-rule*)
by (auto intro!: *R-assign-rule*)
— Then we refine the dynamics.
have *dynup*: ($x' = f(c_i - c_o) \ \& \ G \ hmax(c_i - c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI$
hmin hmax(c_i - c_o))) \leq
rel-R [$\lambda s. s\$4 = 0 \wedge ?Ictrl \ s$] [*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*: ($x' = f(-c_o) \ \& \ G \ hmin(-c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI$
hmin hmax(-c_o))) \leq
rel-R [$\lambda s. s\$4 \neq 0 \wedge ?Ictrl \ s$] [*I 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-R* [?*Ictrl*] [*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-inv*: [*I hmin hmax*] \leq [*I hmin hmax*]
by *simp*
have *inv-pos*: [*I hmin hmax*] \leq [$\lambda s. hmin \leq s\$1 \wedge s\$1 \leq hmax$]
by *simp*
have *inv-inv*: *rel-R* [*I hmin hmax*] [?*Ictrl*]; (*rel-R* [?*Ictrl*] [*I hmin hmax*])
 \leq *rel-R* [*I hmin hmax*] [*I hmin hmax*]
by (rule *R-seq*)
have *loopref*: *LOOP rel-R* [*I hmin hmax*] [?*Ictrl*]; (*rel-R* [?*Ictrl*] [*I hmin*
hmax]) *INV I hmin hmax* \leq ?*ref*
apply(rule *R-loop*)
using *pre-inv inv-inv inv-pos by auto*
have *obs*: ?*ctrl*; ?*dyn* \leq *rel-R* [*I hmin hmax*] [?*Ictrl*]; (*rel-R* [?*Ictrl*] [*I hmin*
hmax])
apply(rule *R-seq-mono*)
using *ctrl dyn by auto*
show *LOOP* (?*ctrl*; ?*dyn*) *INV I 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 kat2ndfun
  imports
    ../hs-prelims-ka
    ../hs-prelims-dyn-sys

```

```
begin
```

0.10.1 Store and Hoare triples

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

— We start by deleting some conflicting notation.

```

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

```

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

```

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

```

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

```

 $\lceil P \rceil \leq \lceil Q \rceil = (\forall s. P\ s \longrightarrow Q\ s)$ 
 $(\lceil P \rceil = \lceil Q \rceil) = (\forall s. P\ s = Q\ s)$ 
 $(\lceil P \rceil \cdot \lceil Q \rceil) = \lceil \lambda s. P\ s \wedge Q\ s \rceil$ 
 $(\lceil P \rceil + \lceil Q \rceil) = \lceil \lambda s. P\ s \vee Q\ s \rceil$ 
 $\text{tt } \lceil P \rceil = \lceil P \rceil$ 
 $n\ \lceil P \rceil = \lceil \lambda s. \neg P\ s \rceil$ 

```

unfolding $p2ndf\text{-}def\ one\text{-}nd\text{-}fun\text{-}def\ less\text{-}eq\text{-}nd\text{-}fun\text{-}def\ times\text{-}nd\text{-}fun\text{-}def\ plus\text{-}nd\text{-}fun\text{-}def$

by $(auto\ simp:\ nd\text{-}fun\text{-}eq\text{-}iff\ kcomp\text{-}def\ le\text{-}fun\text{-}def\ n\text{-}op\text{-}nd\text{-}fun\text{-}def)$

— Meaning of the state-transformer Hoare triple

lemma $ndfun\text{-}kat\text{-}H$: $Hoare\ \lceil P \rceil\ X\ \lceil Q \rceil \longleftrightarrow (\forall s\ s'.\ P\ s \longrightarrow s' \in (X\bullet)\ s \longrightarrow Q\ s')$

unfolding $Hoare\text{-}def\ p2ndf\text{-}def\ less\text{-}eq\text{-}nd\text{-}fun\text{-}def\ times\text{-}nd\text{-}fun\text{-}def\ kcomp\text{-}def$
by $(auto\ simp\ add:\ le\text{-}fun\text{-}def\ n\text{-}op\text{-}nd\text{-}fun\text{-}def)$

— Hoare triple for skip and a simp-rule

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

lemma $H\text{-}skip$: $Hoare\ \lceil P \rceil\ skip\ \lceil P \rceil$
using $H\text{-}skip$ **by** $blast$

lemma $sH\text{-}skip[simp]$: $Hoare\ \lceil P \rceil\ skip\ \lceil Q \rceil \longleftrightarrow \lceil P \rceil \leq \lceil Q \rceil$
unfolding $ndfun\text{-}kat\text{-}H$ **by** $(simp\ add:\ one\text{-}nd\text{-}fun\text{-}def)$

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

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

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

lemma $H\text{-}assign$: $P = (\lambda s.\ Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s))))\ j) \implies Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil$
unfolding $ndfun\text{-}kat\text{-}H\ assign\text{-}def\ vec\text{-}upd\text{-}def$ **by** $force$

lemma $sH\text{-}assign[simp]$: $Hoare\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil \longleftrightarrow (\forall s.\ P\ s \longrightarrow Q\ (\chi\ j.\ (((\$)\ s)(x := (e\ s))))\ j))$
unfolding $ndfun\text{-}kat\text{-}H\ vec\text{-}upd\text{-}def\ assign\text{-}def$ **by** $(auto\ simp:\ fun\text{-}upd\text{-}def)$

— Next, the Hoare rule of the composition

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

lemma $H\text{-}seq$: $Hoare\ \lceil P \rceil\ X\ \lceil R \rceil \implies Hoare\ \lceil R \rceil\ Y\ \lceil Q \rceil \implies Hoare\ \lceil P \rceil\ (X\ ;\ Y)\ \lceil Q \rceil$
by $(auto\ intro:\ H\text{-}seq)$

lemma $sH\text{-}seq$: $Hoare\ \lceil P \rceil\ (X\ ;\ Y)\ \lceil Q \rceil = Hoare\ \lceil P \rceil\ (X)\ \lceil \lambda s.\ \forall s'.\ s' \in (Y\bullet)\ s \longrightarrow Q\ s' \rceil$
unfolding $ndfun\text{-}kat\text{-}H$ **by** $(auto\ simp:\ times\text{-}nd\text{-}fun\text{-}def\ kcomp\text{-}def)$

— Rewriting the Hoare rule for the conditional statement

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

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

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

— Rewriting the Hoare rule for the while loop

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

lemma *sH-while-inv*: $\forall s. P \ s \longrightarrow I \ s \Longrightarrow \forall s. I \ s \wedge \neg B \ s \longrightarrow Q \ s \Longrightarrow \text{Hoare}$
 $[\lambda s. I \ s \wedge B \ s] \ X \ [I]$
 $\Longrightarrow \text{Hoare } [P] \ (\text{WHILE } B \ \text{INV } I \ \text{DO } X) \ [Q]$
by (rule *H-while-inv*, *simp-all add*: *ndfun-kat-H*)

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

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

lemma *H-loop*: $\text{Hoare } [P] \ X \ [P] \Longrightarrow \text{Hoare } [P] \ (\text{LOOP } X \ \text{INV } I) \ [P]$
by (auto *intro*: *H-loop*)

lemma *H-loopI*: $\text{Hoare } [I] \ X \ [I] \Longrightarrow [P] \leq [I] \Longrightarrow [I] \leq [Q] \Longrightarrow \text{Hoare } [P]$
 $(\text{LOOP } X \ \text{INV } I) \ [Q]$
using *H-loop-inv*[of $[P] \ [I] \ X \ [Q]$] **by** auto

0.10.2 Verification of hybrid programs

— Verification by providing evolution

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

lemma *H-g-evol*:
fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$

assumes $P = (\lambda s. (\forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)))$
shows $\text{Hoare } [P] \ (\text{EVOL } \varphi \ G \ T) \ [Q]$
unfolding $\text{ndfun-kat-H } g\text{-evol-def } g\text{-orbit-eq}$ **using** assms **by** clarsimp

lemma sH-g-evol[simp] :
fixes $\varphi :: ('a::\text{preorder}) \Rightarrow 'b \Rightarrow 'b$
shows $\text{Hoare } [P] \ (\text{EVOL } \varphi \ G \ T) \ [Q] = (\forall s. P \ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)))$
unfolding $\text{ndfun-kat-H } g\text{-evol-def } g\text{-orbit-eq}$ **by** auto

— Verification by providing solutions

definition $g\text{-ode} :: ('a::\text{banach}) \Rightarrow 'a \Rightarrow 'a \text{ pred} \Rightarrow \text{real set} \Rightarrow 'a \text{ set} \Rightarrow$
 $\text{real} \Rightarrow 'a \text{ nd-fun } ((1x' = - \ \& \ - \ \text{on} \ - \ - \ @ \ -))$
where $(x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0) \equiv (\lambda s. g\text{-orbital } f \ G \ T \ S \ t_0 \ s)^\bullet$

lemma $H\text{-g-orbital}$:
 $P = (\lambda s. (\forall X \in \text{ivp-sols } (\lambda t. f) \ T \ S \ t_0 \ s. \forall t \in T. (\forall \tau \in \text{down } T \ t. G (X \ \tau)) \longrightarrow Q (X \ t))) \Longrightarrow$
 $\text{Hoare } [P] \ (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0) \ [Q]$
unfolding $\text{ndfun-kat-H } g\text{-ode-def } g\text{-orbital-eq}$ **by** clarsimp

lemma sH-g-orbital : $\text{Hoare } [P] \ (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ t_0) \ [Q] =$
 $(\forall s. P \ s \longrightarrow (\forall X \in \text{ivp-sols } (\lambda t. f) \ T \ S \ t_0 \ s. \forall t \in T. (\forall \tau \in \text{down } T \ t. G (X \ \tau)) \longrightarrow Q (X \ t)))$
unfolding $g\text{-orbital-eq } g\text{-ode-def } \text{ndfun-kat-H}$ **by** auto

context local-flow
begin

lemma $H\text{-g-ode}$:
assumes $P = (\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)))$
shows $\text{Hoare } [P] \ (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ 0) \ [Q]$
proof $(\text{unfold } \text{ndfun-kat-H } g\text{-ode-def } g\text{-orbital-eq } \text{assms}, \text{clarsimp})$
fix $s \ t \ X$
assume $\text{hyps}: t \in T \ \forall x. x \in T \wedge x \leq t \longrightarrow G (X \ x) \ X \in \text{Sols } (\lambda t. f) \ T \ S \ 0 \ s$
and main: $s \in S \longrightarrow (\forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s))$
have $s \in S$
using $\text{ivp-solsD}[OF \ \text{hyps}(3)] \ \text{init-time}$ **by** auto
hence $\forall \tau \in \text{down } T \ t. X \ \tau = \varphi \ \tau \ s$
using eq-solution hyps **by** blast
thus $Q (X \ t)$
using $\text{main } \langle s \in S \rangle \ \text{hyps}$ **by** fastforce
qed

lemma sH-g-ode : $\text{Hoare } [P] \ (x' = f \ \& \ G \ \text{on} \ T \ S \ @ \ 0) \ [Q] =$
 $(\forall s \in S. P \ s \longrightarrow (\forall t \in T. (\forall \tau \in \text{down } T \ t. G (\varphi \ \tau \ s)) \longrightarrow Q (\varphi \ t \ s)))$

proof(*unfold sH-g-orbital, clarsimp, safe*)
fix $s\ t$
assume $hyps: s \in S\ P\ s\ t \in T\ \forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)$
and $main: \forall s. P\ s \longrightarrow (\forall X \in Sols\ (\lambda t. f)\ T\ S\ 0\ s. \forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
hence $(\lambda t. \varphi\ t\ s) \in Sols\ (\lambda t. f)\ T\ S\ 0\ s$
using *in-ivp-sols* **by** *blast*
thus $Q\ (\varphi\ t\ s)$
using *main hyps* **by** *fastforce*
next
fix $s\ X\ t$
assume $hyps: P\ s\ X \in Sols\ (\lambda t. f)\ T\ S\ 0\ s\ t \in T\ \forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (X\ \tau)$
and $main: \forall s \in S. P\ s \longrightarrow (\forall t \in T. (\forall \tau. \tau \in T \wedge \tau \leq t \longrightarrow G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
hence $obs: s \in S$
using *ivp-sols-def*[*of* $\lambda t. f$] *init-time* **by** *auto*
hence $\forall \tau \in down\ T\ t. X\ \tau = \varphi\ \tau\ s$
using *eq-solution hyps* **by** *blast*
thus $Q\ (X\ t)$
using *hyps main obs* **by** *auto*
qed

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

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

hence $\forall \tau \in \text{down } \{0..\tau\} \ t. \ X \ \tau = \varphi \ \tau \ s$
using *obs1 obs2 apply(clarsimp)*
by (*rule eq-solution-ivl*) (*auto simp: closed-segment-eq-real-ivl*)
thus $Q \ (X \ t)$
using *hypos main* $\langle s \in S \rangle$ **by** *auto*
qed

lemma *sH-orbit*: $\text{Hoare } [P] \ (\gamma^{\varphi\bullet}) \ [Q] = (\forall s \in S. \ P \ s \longrightarrow (\forall t \in T. \ Q \ (\varphi \ t \ s)))$
using *sH-g-ode unfolding orbit-def g-ode-def* **by** *auto*

end

— Verification with differential invariants

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

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 ivp-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*: $\text{Hoare } [I] \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0) \ [I] \Longrightarrow [P] \leq [I] \Longrightarrow$
 $[\lambda s. \ I \ s \wedge G \ s] \leq [Q] \Longrightarrow \text{Hoare } [P] \ (x' = f \ \& \ G \text{ on } T \ S \ @ \ t_0 \ \text{DINV } I) \ [Q]$
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*: $(\forall s. \ P \ s \longrightarrow Q \ s) \Longrightarrow 1 \leq \text{Ref } [P] \ [Q]$
by (*auto simp: spec-def ndfun-kat-H one-nd-fun-def*)

— Composition

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

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

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

— Assignment

lemma *R-assign*: $(x ::= e) \leq Ref \ [\lambda s. P \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)] \ [P]$
unfolding *spec-def* **by** (*rule H-assign*, *clarsimp simp: fun-eq-iff fun-upd-def*)

lemma *R-assign-rule*: $(\forall s. P \ s \longrightarrow Q \ (\chi \ j. (((\$) \ s)(x := (e \ s))) \ j)) \implies (x ::= e) \leq Ref \ [P] \ [Q]$
unfolding *sH-assign[symmetric] spec-def* .

lemma *R-assignl*: $P = (\lambda s. R \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)) \implies (x ::= e) ; Ref \ [R] \ [Q] \leq Ref \ [P] \ [Q]$
apply(*rule-tac R=R in R-seq-rule*)
by (*rule-tac R-assign-rule, simp-all*)

lemma *R-assignr*: $R = (\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j)) \implies Ref \ [P] \ [R] ; (x ::= e) \leq Ref \ [P] \ [Q]$
apply(*rule-tac R=R in R-seq-rule, simp*)
by (*rule-tac R-assign-rule, simp*)

lemma $(x ::= e) ; Ref \ [Q] \ [Q] \leq Ref \ [(\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j))] \ [Q]$
by (*rule R-assignl simp*)

lemma $Ref \ [Q] \ [(\lambda s. Q \ (\chi \ j. (((\$) \ s)(x := e \ s)) \ j))] ; (x ::= e) \leq Ref \ [Q] \ [Q]$
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 $(EVOL\ \varphi\ G\ T) \leq Ref\ [\lambda s. \forall t \in T. (\forall \tau \in down\ T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow P\ (\varphi\ t\ s)]\ [P]$
unfolding *spec-def* **by** (*rule* *H-g-evol*, *simp*)

lemma *R-g-evol-rule*:
fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $(\forall s. P\ s \longrightarrow (\forall t \in T. (\forall \tau \in down\ T\ t. G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))) \implies (EVOL\ \varphi\ G\ T) \leq Ref\ [P]\ [Q]$
unfolding *sH-g-evol[symmetric]* *spec-def* .

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

$s)) \longrightarrow Q(\varphi \ t \ s)] \upharpoonright [Q]$
by (rule *R-g-evoll*) *simp*

lemma

fixes $\varphi :: ('a::preorder) \Rightarrow 'b \Rightarrow 'b$
shows $Ref \upharpoonright [Q] \upharpoonright [\lambda s. \forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow Q(\varphi \ t \ s)]; EVOL$
 $\varphi \ G \ T \leq Ref \upharpoonright [Q] \upharpoonright [Q]$
by (rule *R-g-evolr*) *simp*

— Evolution command (ode)

context *local-flow*

begin

lemma *R-g-ode*: $(x' = f \ \& \ G \ on \ T \ S \ @ \ 0) \leq Ref \upharpoonright [\lambda s. s \in S \longrightarrow (\forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow P(\varphi \ t \ s))] \upharpoonright [P]$
unfolding *spec-def* **by** (rule *H-g-ode*, *simp*)

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

lemma *R-g-odel*: $P = (\lambda s. \forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow R(\varphi \ t \ s)) \Longrightarrow$
 $(x' = f \ \& \ G \ on \ T \ S \ @ \ 0) ; Ref \upharpoonright [R] \upharpoonright [Q] \leq Ref \upharpoonright [P] \upharpoonright [Q]$
apply(rule-tac *R=R in R-seq-rule*)
by (rule-tac *R-g-ode-rule*, *simp-all*)

lemma *R-g-oder*: $R = (\lambda s. \forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow Q(\varphi \ t \ s)) \Longrightarrow$

$Ref \upharpoonright [P] \upharpoonright [R]; (x' = f \ \& \ G \ on \ T \ S \ @ \ 0) \leq Ref \upharpoonright [P] \upharpoonright [Q]$
apply(rule-tac *R=R in R-seq-rule*, *simp*)
by (rule-tac *R-g-ode-rule*, *simp*)

lemma $(x' = f \ \& \ G \ on \ T \ S \ @ \ 0) ; Ref \upharpoonright [Q] \upharpoonright [Q] \leq Ref \upharpoonright [\lambda s. \forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow Q(\varphi \ t \ s)] \upharpoonright [Q]$
by (rule *R-g-odel*) *simp*

lemma $Ref \upharpoonright [Q] \upharpoonright [\lambda s. \forall t \in T. (\forall \tau \in down \ T \ t. G(\varphi \ \tau \ s)) \longrightarrow Q(\varphi \ t \ s)]; (x' = f \ \& \ G \ on \ T \ S \ @ \ 0) \leq Ref \upharpoonright [Q] \upharpoonright [Q]$
by (rule *R-g-oder*) *simp*

lemma *R-g-ode-ivl*:

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

end

— Evolution command (invariants)

lemma *R-g-ode-inv: diff-invariant* $I f T S t_0 G \implies [P] \leq [I] \implies [\lambda s. I s \wedge G s] \leq [Q] \implies$
 $(x' = f \ \& \ G \text{ on } T S @ t_0 \text{ DINV } I) \leq \text{Ref } [P] [Q]$
unfolding *spec-def by (auto simp: H-g-ode-inv)*

0.10.4 Derivation of the rules of dL

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

lemma *diff-solve-axiom:*

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

lemma *diff-solve-rule:*

assumes *local-flow* $f T \text{ UNIV } \varphi$
and $\forall s. P s \longrightarrow (\forall t \in T. (\mathcal{P} (\lambda t. \varphi t s) (\text{down } T t) \subseteq \{s. G s\}) \longrightarrow Q (\varphi t s))$
shows *Hoare* $[P] (x' = f \ \& \ G \text{ on } T \text{ UNIV } @ 0) [Q]$
using *assms by (subst local-flow.sH-g-ode, 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 *Thyp: is-interval* $T t_0 \in T$
and *wp-C:Hoare* $[P] (x' = f \ \& \ G \text{ on } T S @ t_0) [C]$
and *wp-Q:Hoare* $[P] (x' = f \ \& \ (\lambda s. G s \wedge C s) \text{ on } T S @ t_0) [Q]$
shows *Hoare* $[P] (x' = f \ \& \ G \text{ on } T S @ t_0) [Q]$
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** $t \in T$
and $x\text{-ivp}: X \in \text{ivp-sols } (\lambda t. f) T S t_0 s$
and *guard-x:* $\forall x. x \in T \wedge x \leq t \longrightarrow G (X x)$
have $\forall t \in (\text{down } T t). X t \in \text{g-orbital } f G T S t_0 s$
using *g-orbitalI[OF x-ivp] guard-x by auto*
hence $\forall t \in (\text{down } T t). C (X t)$
using *wp-C <P s> by (subst (asm) ndfun-kat-H, auto simp: g-ode-def)*
hence $X t \in \text{g-orbital } f (\lambda s. G s \wedge C s) T S t_0 s$
using *guard-x <t ∈ T> 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

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

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

end

```

0.10.5 Examples

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

```

theory kat2ndfun-examples
imports kat2ndfun

```

```

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

abbreviation pend-flow :: real  $\Rightarrow$  real2  $\Rightarrow$  real2 ( $\varphi$ )
where  $\varphi$   $\tau$  s  $\equiv$  ( $\chi$  i. if i = 1 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: 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* φ
apply(*unfold-locales*, *simp-all* *add*: *local-lipschitz-def* *lipschitz-on-def* *vec-eq-iff*,
clarsimp)
apply(*rule-tac* $x=1$ **in** *exI*, *clarsimp*, *rule-tac* $x=1$ **in** *exI*)
apply(*simp* *add*: *dist-norm* *norm-vec-def* *L2-set-def* *power2-commute* *UNIV-2*)
by (*auto simp: forall-2 intro!: poly-derivatives*)

lemma *pendulum-flow*: *Hoare* $\lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil (x'=f \ \& \ G) \lceil \lambda s. r^2 = (s\$1)^2 + (s\$2)^2 \rceil$
by (*simp only: local-flow.sH-g-ode[OF local-flow-pend], simp*)

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

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

— Verified with differential invariants

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

lemma [*bb-real-arith*]:
assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
shows $(x::\text{real}) \leq h$
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* :: *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 (f g) UNIV UNIV 0 G*
 unfolding *dinv* apply (rule *diff-invariant-rules*, *simp*, *simp*, *clarify*)
 by (*auto intro!*: *poly-derivatives*)

lemma *bouncing-ball-inv*: $g < 0 \implies h \geq 0 \implies \text{Hoare}$

$[\lambda s. s\$1 = h \wedge s\$2 = 0]$
 (*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 $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$
 $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs = ?rhs*)
proof–
have *?lhs* = $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
apply(*subst Rat.sign-simps(18)*) +
by(*auto simp: semiring-normalization-rules(29)*)
also have $\dots = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot h + v \cdot v$ (**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*: $g < 0 \implies h \geq 0 \implies \text{Hoare}$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0]$
(*LOOP*
 $((\text{EVOL } (\varphi \ g) \ (\lambda s. s\$1 \geq 0) \ T);$
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$
apply(*rule H-loopI, rule H-seq[where R= $\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot$*
 $h + s\$2 \cdot s\$2]$)
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*: $g < 0 \implies h \geq 0 \implies \text{Hoare}$
 $[\lambda s. s\$1 = h \wedge s\$2 = 0]$
(*LOOP*
 $((x' = f \ g \ \& \ (\lambda s. s\$1 \geq 0));$
 $(\text{IF } (\lambda s. s\$1 = 0) \ \text{THEN } (2 ::= (\lambda s. - s\$2)) \ \text{ELSE skip}))$
 $\text{INV } (\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2)$
 $) \ [\lambda s. 0 \leq s\$1 \wedge s\$1 \leq h]$

```

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(subst local-flow.sH-g-ode[OF local-flow-ball])
      apply(force simp: bb-real-arith)
    by (rule H-cond) (auto simp: bb-real-arith)

```

— Refined with annotated dynamics

```

lemma R-bb-assign:  $g < (0::real) \implies 0 \leq h \implies$ 
   $2 ::= (\lambda s. - s\$2) \leq Ref$ 
   $[\lambda s. s\$1 = 0 \wedge 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$ 
   $[\lambda s. 0 \leq s\$1 \wedge 2 \cdot g \cdot s\$1 = 2 \cdot g \cdot h + s\$2 \cdot s\$2]$ 
  by (rule R-assign-rule, auto)

```

```

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

the thermostat keeps the room's temperature between $Tmin$ and $Tmax$.

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

abbreviation *therm-guard* :: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow real^4 \Rightarrow bool$ (G)
where $G \ Tmin \ Tmax \ a \ L \ s \equiv (s\$2 \leq -(\ln((L - (\text{if } L=0 \text{ then } Tmin \text{ else } Tmax)) / (L - s\$3)))) / a$

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

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

— 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-lsqrt*) *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 *thyps*: $0 \leq (\tau::real) \ \forall \tau \in \{0.. \tau\}. \tau \leq -(\ln(Tmin / T) / a)$

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

proof–

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*: $Tmin \leq T \leq Tmax$ $Tmax < (L::real)$
 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 $H\text{-}g\text{-ode-therm} = local\text{-}flow.sH\text{-}g\text{-ode-ivl}[OF\ local\text{-}flow\text{-}therm - UNIV-I]$

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 ( $\lambda s. s\$4 = 0 \wedge s\$3 \leq T_{min} + 1$ ) THEN
    (4 ::= ( $\lambda s. 1$ ))
    ELSE IF ( $\lambda s. s\$4 = 1 \wedge s\$3 \geq T_{max} - 1$ ) THEN
      (4 ::= ( $\lambda s. 0$ ))
    ELSE skip);
  — dynamics
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = f \ a \ 0 \ \& \ G \ T_{min} \ T_{max} \ a \ 0 \ \text{on} \ \{0..\tau\} \ \text{UNIV} \ @ \ 0$ )
  ELSE
    ( $x' = f \ a \ L \ \& \ G \ T_{min} \ T_{max} \ a \ L \ \text{on} \ \{0..\tau\} \ \text{UNIV} \ @ \ 0$ ))
) INV I  $T_{min} \ T_{max}$ )
[I  $T_{min} \ T_{max}$ ]
apply(rule H-loopI)
apply(rule-tac  $R = \lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$  in H-seq)
apply(rule-tac  $R = \lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0$  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  $T_{min}$ ]
and therm-dyn-down-real-arith[OF assms(1,3), of -  $T_{max}$ ] by auto

```

— Refined with the flow

lemma *R-therm-dyn-down*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < T_{min}$  and  $T_{max} < L$ 
shows Ref [ $\lambda s. s\$4 = 0 \wedge I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I  $T_{min} \ T_{max}$ ]  $\geq$ 
  ( $x' = f \ a \ 0 \ \& \ G \ T_{min} \ T_{max} \ a \ 0 \ \text{on} \ \{0..\tau\} \ \text{UNIV} \ @ \ 0$ )
apply(rule local-flow.R-g-ode-ivl[OF local-flow-therm])
using assms therm-dyn-down-real-arith[OF assms(1,3), of -  $T_{max}$ ] by auto

```

lemma *R-therm-dyn-up*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < T_{min}$  and  $T_{max} < L$ 
shows Ref [ $\lambda s. s\$4 \neq 0 \wedge I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I  $T_{min} \ T_{max}$ ]  $\geq$ 
  ( $x' = f \ a \ L \ \& \ G \ T_{min} \ T_{max} \ a \ L \ \text{on} \ \{0..\tau\} \ \text{UNIV} \ @ \ 0$ )
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  $T_{min}$ ] by
auto

```

lemma *R-therm-dyn*:

```

assumes  $a > 0$  and  $0 \leq \tau$  and  $0 < T_{min}$  and  $T_{max} < L$ 
shows Ref [ $\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1$ ] [I  $T_{min} \ T_{max}$ ]  $\geq$ 
  (IF ( $\lambda s. s\$4 = 0$ ) THEN
    ( $x' = f \ a \ 0 \ \& \ G \ T_{min} \ T_{max} \ a \ 0 \ \text{on} \ \{0..\tau\} \ \text{UNIV} \ @ \ 0$ )
  ELSE

```

$(x' = f \ a \ L \ \& \ G \ T_{min} \ T_{max} \ a \ L \ \text{on } \{0..\tau\} \ \text{UNIV} \ @ \ 0))$
apply(rule order-trans, rule R-cond-mono)
using R-therm-dyn-down[OF assms] R-therm-dyn-up[OF assms] **by** (auto intro!: R-cond)

lemma R-therm-assign1: Ref $[I \ T_{min} \ T_{max}] \ [\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0]$
 $\geq (2 ::= (\lambda s. 0))$
by (auto simp: R-assign-rule)

lemma R-therm-assign2:
 Ref $[\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0] \ [\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1]$
 $\geq (3 ::= (\lambda s. s\$1))$
by (auto simp: R-assign-rule)

lemma R-therm-ctrl:
 Ref $[I \ T_{min} \ T_{max}] \ [\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \geq$
 $(2 ::= (\lambda s. 0));$
 $(3 ::= (\lambda s. s\$1));$
 $(IF \ (\lambda s. s\$4 = 0 \wedge s\$3 \leq T_{min} + 1) \ THEN$
 $(4 ::= (\lambda s. 1))$
 $ELSE \ IF \ (\lambda s. s\$4 = 1 \wedge s\$3 \geq T_{max} - 1) \ THEN$
 $(4 ::= (\lambda s. 0))$
 $ELSE \ skip)$
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: Ref $[I \ T_{min} \ T_{max}] \ [I \ T_{min} \ T_{max}] \geq$
 $(LOOP$
 $Ref \ [I \ T_{min} \ T_{max}] \ [\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1];$
 $Ref \ [\lambda s. I \ T_{min} \ T_{max} \ s \wedge s\$2 = 0 \wedge s\$3 = s\$1] \ [I \ T_{min} \ T_{max}]$
 $INV \ I \ T_{min} \ T_{max})$
by (intro R-loop R-seq, simp-all)

lemma R-thermostat-flow:
assumes $a > 0$ **and** $0 \leq \tau$ **and** $0 < T_{min}$ **and** $T_{max} < L$
shows Ref $[I \ T_{min} \ T_{max}] \ [I \ T_{min} \ T_{max}] \geq$
 $(LOOP$ (
 — control

```

(2 ::= (λs. 0));(3 ::= (λs. s$1));
(IF (λs. s$4 = 0 ∧ s$3 ≤ Tmin + 1) THEN
  (4 ::= (λs.1))
  ELSE IF (λs. s$4 = 1 ∧ s$3 ≥ Tmax - 1) THEN
    (4 ::= (λs.0))
    ELSE skip);
— dynamics
(IF (λs. s$4 = 0) THEN
  (x' = f a 0 & G Tmin Tmax a 0 on {0..τ} UNIV @ 0)
  ELSE
    (x' = f a L & G Tmin Tmax a L on {0..τ} 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* (*φ*)
and *therm-guard* (*G*)
and *therm-loop-inv* (*I*)

Water tank

— Variation of Hespanha and [?]

abbreviation *tank-vec-field* :: *real* ⇒ *real*^4 ⇒ *real*^4 (*f*)
where *f k s* ≡ (χ *i*. if *i* = 2 then 1 else (if *i* = 1 then *k* else 0))

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

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

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

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

— Verified with the flow

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

lemma *tank-arith*:

assumes $0 \leq (\tau::\text{real})$ **and** $0 < c_o$ **and** $c_o < c_i$
shows $\forall \tau \in \{0..\tau\}. \tau \leq -((hmin - y) / c_o) \implies hmin \leq y - c_o * \tau$
and $\forall \tau \in \{0..\tau\}. \tau \leq (hmax - y) / (c_i - c_o) \implies (c_i - c_o) * \tau + y \leq hmax$
and $hmin \leq y \implies hmin \leq (c_i - c_o) \cdot \tau + y$
and $y \leq hmax \implies y - c_o \cdot \tau \leq hmax$
apply(*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
 $((2 ::= (\lambda s. 0)); (3 ::= (\lambda s. s\$1)));$
 (*IF* ($\lambda s. s\$4 = 0 \wedge s\$3 \leq hmin + 1$) *THEN* ($4 ::= (\lambda s. 1)$) *ELSE*
 (*IF* ($\lambda s. s\$4 = 1 \wedge s\$3 \geq hmax - 1$) *THEN* ($4 ::= (\lambda s. 0)$) *ELSE skip*));
 — dynamics
 (*IF* ($\lambda s. s\$4 = 0$) *THEN* ($x' = f(c_i - c_o) \ \& \ G \ hmax \ (c_i - c_o)$ on $\{0..\tau\}$ *UNIV*
 @ 0)
 ELSE ($x' = f(-c_o) \ \& \ G \ hmin \ (-c_o)$ on $\{0..\tau\}$ *UNIV* @ 0)))
INV I hmin hmax) [*I hmin hmax*]
apply(*rule H-loopI*)
apply(*rule-tac R = $\lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\1 in H-seq*)
apply(*rule-tac R = $\lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0 \wedge s\$3 = s\1 in H-seq*)
apply(*rule-tac R = $\lambda s. I \ hmin \ hmax \ s \wedge s\$2 = 0$ in H-seq, simp, simp*)
apply(*rule H-cond, simp-all add: H-g-ode-tank[OF assms(1)]*)
using *assms tank-arith[OF - assms(2,3)]* **by** *auto*

— Verified with differential invariants

lemma *tank-diff-inv*:

$0 \leq \tau \implies \text{diff-invariant } (dI \ hmin \ hmax \ k) \ (f \ k) \ \{0..\tau\} \ \text{UNIV } 0 \ \text{Guard}$
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*)
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** $b: hmin \leq y_0$ **and** $g: \tau \leq (hmax - y_0) / (c_i - c_o)$
shows $hmin \leq (c_i - c_o) \cdot \tau + y_0$ **and** $(c_i - c_o) \cdot \tau + y_0 \leq hmax$
proof —

```

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 linordered-field-class.sign-simps(39) mult-less-cancel-right)

```

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' = f (ci - co) & G hmax (ci - co) on {0..τ} UNIV @ 0 DINV (dI hmin
hmax (ci - co)))
      ELSE
      (x' = f (-co) & G hmin (-co) on {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 - -co 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 $\text{Ref } [I \text{ hmin hmax}] [I \text{ hmin hmax}] \geq$
 $(\text{LOOP}$
 — control
 $((\mathcal{Q} ::= (\lambda s. 0)); (\mathcal{Q} ::= (\lambda s. s\$1)));$
 $(\text{IF } (\lambda s. s\$4 = 0 \wedge s\$3 \leq \text{hmin} + 1) \text{ THEN } (\mathcal{Q} ::= (\lambda s. 1)) \text{ ELSE}$
 $(\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{hmax} - 1) \text{ THEN } (\mathcal{Q} ::= (\lambda s. 0)) \text{ ELSE skip}));$
 — dynamics
 $(\text{IF } (\lambda s. s\$4 = 0) \text{ THEN}$
 $(x' = f(c_i - c_o) \ \& \ G \text{ hmax } (c_i - c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \text{ hmin}$
 $\text{hmax } (c_i - c_o)))$
 ELSE
 $(x' = f(-c_o) \ \& \ G \text{ hmin } (-c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI \text{ hmin hmax}$
 $(-c_o))))$
 $\text{INV } I \text{ hmin hmax}) \text{ (is LOOP } (?ctrl; ?dyn) \text{ INV - } \leq ?ref)$

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 $?cond = \lambda s. s\$4 = 0 \wedge s\$3 \leq \text{hmin} + 1$
have $\text{ifbranch1: } \mathcal{Q} ::= (\lambda s. 1) \leq \text{Ref } [\lambda s. ?cond \ s \wedge ?Ictrl \ s] [?Ictrl] \text{ (is - } \leq$
 $?branch1)$
by (rule *R-assign-rule, simp*)
have $\text{ifbranch2: } (\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{hmax} - 1) \text{ THEN } (\mathcal{Q} ::= (\lambda s. 0))$
 $\text{ELSE skip}) \leq$
 $\text{Ref } [\lambda s. \neg ?cond \ s \wedge ?Ictrl \ s] [?Ictrl] \text{ (is - } \leq ?branch2)$
apply(rule *order-trans, rule R-cond-mono*) **defer defer**
by (rule *R-cond*) (auto intro!: *R-assign-rule R-skip*)
have $\text{ifthenelse: } (\text{IF } ?cond \text{ THEN } ?branch1 \text{ ELSE } ?branch2) \leq \text{Ref } [?Ictrl]$
 $[?Ictrl] \text{ (is ifthenelse } \leq -)$
by (rule *R-cond*)
have $(\text{IF } ?cond \text{ THEN } (\mathcal{Q} ::= (\lambda s. 1)) \text{ ELSE } (\text{IF } (\lambda s. s\$4 = 1 \wedge s\$3 \geq \text{hmax}$
 $- 1) \text{ THEN } (\mathcal{Q} ::= (\lambda s. 0)) \text{ ELSE skip})) \leq$
 $\text{Ref } [?Ictrl] [?Ictrl]$
apply(rule-tac $y = ?ifthenelse$ **in** *order-trans, rule R-cond-mono*)
using $\text{ifbranch1 ifbranch2 ifthenelse by auto}$
hence $ctrl: ?ctrl \leq \text{Ref } [I \text{ hmin hmax}] [?Ictrl]$
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 $\text{dynup: } (x' = f(c_i - c_o) \ \& \ G \text{ hmax } (c_i - c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI$
 $\text{hmin hmax } (c_i - c_o))) \leq$
 $\text{Ref } [\lambda s. s\$4 = 0 \wedge ?Ictrl \ s] [I \text{ hmin hmax}]$
apply(rule *R-g-ode-inv[OF tank-diff-inv[OF assms(1)]]*)
using $\text{assms by (auto simp: tank-inv-arith1)}$
have $\text{dyndown: } (x' = f(-c_o) \ \& \ G \text{ hmin } (-c_o) \text{ on } \{0..\tau\} \text{ UNIV @ } 0 \text{ DINV } (dI$

```

hmin hmax (-c_o))) ≤
  Ref [λs. s$4 ≠ 0 ∧ ?Icntrl s] [I 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 ≤ 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] ≤ [I hmin hmax]
  by simp
have inv-inv: Ref [I hmin hmax] [?Icntrl]; (Ref [?Icntrl] [I hmin hmax]) ≤
Ref [I hmin hmax] [I hmin hmax]
  by (rule R-seq)
have loopref: LOOP Ref [I hmin hmax] [?Icntrl]; (Ref [?Icntrl] [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] [?Icntrl]; (Ref [?Icntrl] [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 Kleene Algebras

```

theory Kleene-Algebra
imports Kleene-Algebra.Conway
begin

```

0.11.1 Left Near Kleene Algebras

Extending the hierarchy developed in *Kleene-Algebra.Dioid* we now add an operation of Kleene star, finite iteration, or reflexive transitive closure to variants of Dioids. Since a multiplicative unit is needed for defining the star we only consider variants with 1; 0 can be added separately. We consider the left star induction axiom and the right star induction axiom independently since in some applications, e.g., Salomaa's axioms, probabilistic Kleene alge-

bras, or completeness proofs with respect to the equational theory of regular expressions and regular languages, the right star induction axiom is not needed or not valid.

We start with near dioids, then consider pre-dioids and finally dioids. It turns out that many of the known laws of Kleene algebras hold already in these more general settings. In fact, all our equational theorems have been proved within left Kleene algebras, as expected.

Although most of the proofs in this file could be fully automated by Sledgehammer and Metis, we display step-wise proofs as they would appear in a text book. First, this file may then be useful as a reference manual on Kleene algebra. Second, it is better protected against changes in the underlying theories and supports easy translation of proofs into other settings.

```
class left-near-kleene-algebra = near-dioid-one + star-op +
  assumes star-unfoldl:  $1 + x \cdot x^* \leq x^*$ 
  and star-inductl:  $z + x \cdot y \leq y \implies x^* \cdot z \leq y$ 
```

begin

First we prove two immediate consequences of the unfold axiom. The first one states that starred elements are reflexive.

```
lemma star-ref [simp]:  $1 \leq x^*$ 
using star-unfoldl by auto
```

Reflexivity of starred elements implies, by definition of the order, that 1 is an additive unit for starred elements.

```
lemma star-plus-one [simp]:  $1 + x^* = x^*$ 
using less-eq-def star-ref by blast
```

```
lemma star-1l [simp]:  $x \cdot x^* \leq x^*$ 
using star-unfoldl by auto
```

```
lemma  $x^* \cdot x \leq x^*$ 
```

oops

```
lemma  $x \cdot x^* = x^*$ 
```

oops

Next we show that starred elements are transitive.

```
lemma star-trans-eq [simp]:  $x^* \cdot x^* = x^*$ 
```

proof (rule antisym) — this splits an equation into two inequalities

```
have  $x^* + x \cdot x^* \leq x^*$ 
```

```
by auto
```

```
thus  $x^* \cdot x^* \leq x^*$ 
```

```
by (simp add: star-inductl)
```

```

    next show  $x^* \leq x^* \cdot x^*$ 
      using mult-isor star-ref by fastforce
qed

```

```

lemma star-trans:  $x^* \cdot x^* \leq x^*$ 
  by simp

```

We now derive variants of the star induction axiom.

```

lemma star-inductl-var:  $x \cdot y \leq y \implies x^* \cdot y \leq y$ 
proof -
  assume  $x \cdot y \leq y$ 
  hence  $y + x \cdot y \leq y$ 
    by simp
  thus  $x^* \cdot y \leq y$ 
    by (simp add: star-inductl)
qed

```

```

lemma star-inductl-var-equiv [simp]:  $x^* \cdot y \leq y \longleftrightarrow x \cdot y \leq y$ 
proof
  assume  $x \cdot y \leq y$ 
  thus  $x^* \cdot y \leq y$ 
    by (simp add: star-inductl-var)
next
  assume  $x^* \cdot y \leq y$ 
  hence  $x^* \cdot y = y$ 
    by (metis eq-iff mult-1-left mult-isor star-ref)
  moreover hence  $x \cdot y = x \cdot x^* \cdot y$ 
    by (simp add: mult.assoc)
  moreover have  $\dots \leq x^* \cdot y$ 
    by (metis mult-isor star-1l)
  ultimately show  $x \cdot y \leq y$ 
    by auto
qed

```

```

lemma star-inductl-var-eq:  $x \cdot y = y \implies x^* \cdot y \leq y$ 
  by (metis eq-iff star-inductl-var)

```

```

lemma star-inductl-var-eq2:  $y = x \cdot y \implies y = x^* \cdot y$ 
proof -
  assume hyp:  $y = x \cdot y$ 
  hence  $y \leq x^* \cdot y$ 
    using mult-isor star-ref by fastforce
  thus  $y = x^* \cdot y$ 
    using hyp eq-iff by auto
qed

```

```

lemma  $y = x \cdot y \longleftrightarrow y = x^* \cdot y$ 

```

oops

lemma $x^* \cdot z \leq y \implies z + x \cdot y \leq y$

oops

lemma *star-inductl-one*: $1 + x \cdot y \leq y \implies x^* \leq y$
using *star-inductl* **by** *force*

lemma *star-inductl-star*: $x \cdot y^* \leq y^* \implies x^* \leq y^*$
by (*simp add: star-inductl-one*)

lemma *star-inductl-eq*: $z + x \cdot y = y \implies x^* \cdot z \leq y$
by (*simp add: star-inductl*)

We now prove two facts related to 1.

lemma *star-subid*: $x \leq 1 \implies x^* = 1$

proof –

assume $x \leq 1$

hence $1 + x \cdot 1 \leq 1$

by *simp*

hence $x^* \leq 1$

by (*metis mult-oner star-inductl*)

thus $x^* = 1$

by (*simp add: order.antisym*)

qed

lemma *star-one* [*simp*]: $1^* = 1$

by (*simp add: star-subid*)

We now prove a subdistributivity property for the star (which is equivalent to isotonicity of star).

lemma *star-subdist*: $x^* \leq (x + y)^*$

proof –

have $x \cdot (x + y)^* \leq (x + y) \cdot (x + y)^*$

by *simp*

also have $\dots \leq (x + y)^*$

by (*metis star-1l*)

thus *?thesis*

using *calculation order-trans star-inductl-star* **by** *blast*

qed

lemma *star-subdist-var*: $x^* + y^* \leq (x + y)^*$

using *join.sup-commute star-subdist* **by** *force*

lemma *star-iso* [*intro*]: $x \leq y \implies x^* \leq y^*$

by (*metis less-eq-def star-subdist*)

We now prove some more simple properties.

lemma *star-invol* [*simp*]: $(x^*)^* = x^*$

```

proof (rule antisym)
  have  $x^* \cdot x^* = x^*$ 
    by (fact star-trans-eq)
  thus  $(x^*)^* \leq x^*$ 
    by (simp add: star-inductl-star)
  have  $(x^*)^* \cdot (x^*)^* \leq (x^*)^*$ 
    by (fact star-trans)
  hence  $x \cdot (x^*)^* \leq (x^*)^*$ 
    by (meson star-inductl-var-equiv)
  thus  $x^* \leq (x^*)^*$ 
    by (simp add: star-inductl-star)
qed

```

```

lemma star2 [simp]:  $(1 + x)^* = x^*$ 
proof (rule antisym)
  show  $x^* \leq (1 + x)^*$ 
    by auto
  have  $x^* + x \cdot x^* \leq x^*$ 
    by simp
  thus  $(1 + x)^* \leq x^*$ 
    by (simp add: star-inductl-star)
qed

```

```

lemma  $1 + x^* \cdot x \leq x^*$ 

```

```

  oops

```

```

lemma  $x \leq x^*$ 

```

```

  oops

```

```

lemma  $x^* \cdot x \leq x^*$ 

```

```

  oops

```

```

lemma  $1 + x \cdot x^* = x^*$ 

```

```

  oops

```

```

lemma  $x \cdot z \leq z \cdot y \implies x^* \cdot z \leq z \cdot y^*$ 

```

```

  oops

```

The following facts express inductive conditions that are used to show that $(x + y)^*$ is the greatest term that can be built from x and y .

```

lemma prod-star-closure:  $x \leq z^* \implies y \leq z^* \implies x \cdot y \leq z^*$ 

```

```

proof –

```

```

  assume assm:  $x \leq z^* \ y \leq z^*$ 

```

```

  hence  $y + z^* \cdot z^* \leq z^*$ 

```



```

    by simp
  hence  $z^* \cdot y \leq z^*$ 
    by (simp add: star-inductl)
  also have  $x \cdot y \leq z^* \cdot y$ 
    by (simp add: assm mult-isor)
  thus  $x \cdot y \leq z^*$ 
    using calculation order.trans by blast
qed

```

```

lemma star-star-closure:  $x^* \leq z^* \implies (x^*)^* \leq z^*$ 
  by (metis star-invol)

```

```

lemma star-closed-unfold:  $x^* = x \implies x = 1 + x \cdot x$ 
  by (metis star-plus-one star-trans-eq)

```

```

lemma  $x^* = x \longleftrightarrow x = 1 + x \cdot x$ 

```

```

oops

```

```

end

```

0.11.2 Left Pre-Kleene Algebras

```

class left-pre-kleene-algebra = left-near-kleene-algebra + pre-dioid-one

```

```

begin

```

We first prove that the star operation is extensive.

```

lemma star-ext [simp]:  $x \leq x^*$ 
proof -
  have  $x \leq x \cdot x^*$ 
    by (metis mult-one mult-isol star-ref)
  thus ?thesis
    by (metis order-trans star-1l)
qed

```

We now prove a right star unfold law.

```

lemma star-1r [simp]:  $x^* \cdot x \leq x^*$ 
proof -
  have  $x + x \cdot x^* \leq x^*$ 
    by simp
  thus ?thesis
    by (fact star-inductl)
qed

```

```

lemma star-unfoldr:  $1 + x^* \cdot x \leq x^*$ 
  by simp

```

```

lemma  $1 + x^* \cdot x = x^*$ 

```

oops

Next we prove a simulation law for the star. It is instrumental in proving further properties.

lemma *star-sim1*: $x \cdot z \leq z \cdot y \implies x^* \cdot z \leq z \cdot y^*$

proof –

assume $x \cdot z \leq z \cdot y$
 hence $x \cdot z \cdot y^* \leq z \cdot y \cdot y^*$
 by (*simp add: mult-isol*)
 also have $\dots \leq z \cdot y^*$
 by (*simp add: mult-isol mult-assoc*)
 finally have $x \cdot z \cdot y^* \leq z \cdot y^*$
 by *simp*
 hence $z + x \cdot z \cdot y^* \leq z \cdot y^*$
 by (*metis join.sup-least mult-isol mult-oner star-ref*)
 thus $x^* \cdot z \leq z \cdot y^*$
 by (*simp add: star-inductl mult-assoc*)

qed

The next lemma is used in omega algebras to prove, for instance, Bachmair and Dershowitz’s separation of termination theorem [?]. The property at the left-hand side of the equivalence is known as *quasicommutation*.

lemma *quasicomm-var*: $y \cdot x \leq x \cdot (x + y)^* \iff y^* \cdot x \leq x \cdot (x + y)^*$

proof

assume $y \cdot x \leq x \cdot (x + y)^*$
 thus $y^* \cdot x \leq x \cdot (x + y)^*$
 using *star-sim1* **by** *force*
next
 assume $y^* \cdot x \leq x \cdot (x + y)^*$
 thus $y \cdot x \leq x \cdot (x + y)^*$
 by (*meson mult-isol order-trans star-ext*)

qed

lemma *star-slide1*: $(x \cdot y)^* \cdot x \leq x \cdot (y \cdot x)^*$

by (*simp add: mult-assoc star-sim1*)

lemma $(x \cdot y)^* \cdot x = x \cdot (y \cdot x)^*$

oops

lemma *star-slide-var1*: $x^* \cdot x \leq x \cdot x^*$

by (*simp add: star-sim1*)

We now show that the (left) star unfold axiom can be strengthened to an equality.

lemma *star-unfoldl-eq* [*simp*]: $1 + x \cdot x^* = x^*$

proof (*rule antisym*)

```

show  $1 + x \cdot x^* \leq x^*$ 
  by (fact star-unfoldl)
have  $1 + x \cdot (1 + x \cdot x^*) \leq 1 + x \cdot x^*$ 
  by (meson join.sup-mono eq-refl mult-isol star-unfoldl)
thus  $x^* \leq 1 + x \cdot x^*$ 
  by (simp add: star-inductl-one)
qed

```

```

lemma  $1 + x^* \cdot x = x^*$ 

```

oops

Next we relate the star and the reflexive transitive closure operation.

```

lemma star-rtc1-eq [simp]:  $1 + x + x^* \cdot x^* = x^*$ 
  by (simp add: join.sup.absorb2)

```

```

lemma star-rtc1:  $1 + x + x^* \cdot x^* \leq x^*$ 
  by simp

```

```

lemma star-rtc2:  $1 + x \cdot x \leq x \longleftrightarrow x = x^*$ 
proof
  assume  $1 + x \cdot x \leq x$ 
  thus  $x = x^*$ 
    by (simp add: local.eq-iff local.star-inductl-one)
next
  assume  $x = x^*$ 
  thus  $1 + x \cdot x \leq x$ 
    using local.star-closed-unfold by auto
qed

```

```

lemma star-rtc3:  $1 + x \cdot x = x \longleftrightarrow x = x^*$ 
  by (metis order-refl star-plus-one star-rtc2 star-trans-eq)

```

```

lemma star-rtc-least:  $1 + x + y \cdot y \leq y \implies x^* \leq y$ 
proof –
  assume  $1 + x + y \cdot y \leq y$ 
  hence  $1 + x \cdot y \leq y$ 
    by (metis join.le-sup-iff mult-isol-var star-trans-eq star-rtc2)
  thus  $x^* \leq y$ 
    by (simp add: star-inductl-one)
qed

```

```

lemma star-rtc-least-eq:  $1 + x + y \cdot y = y \implies x^* \leq y$ 
  by (simp add: star-rtc-least)

```

```

lemma  $1 + x + y \cdot y \leq y \longleftrightarrow x^* \leq y$ 

```

oops

The next lemmas are again related to closure conditions

lemma *star-subdist-var-1*: $x \leq (x + y)^*$
by (*meson join.sup.boundedE star-ext*)

lemma *star-subdist-var-2*: $x \cdot y \leq (x + y)^*$
by (*meson join.sup.boundedE prod-star-closure star-ext*)

lemma *star-subdist-var-3*: $x^* \cdot y^* \leq (x + y)^*$
by (*simp add: prod-star-closure star-iso*)

We now prove variants of sum-elimination laws under a star. These are also known as denesting laws or as sum-star laws.

lemma *star-denest [simp]*: $(x + y)^* = (x^* \cdot y^*)^*$

proof (*rule antisym*)

have $x + y \leq x^* \cdot y^*$

by (*metis join.sup.bounded-iff mult-1-right mult-isol-var mult-onel star-ref star-ext*)

thus $(x + y)^* \leq (x^* \cdot y^*)^*$

by (*fact star-iso*)

have $x^* \cdot y^* \leq (x + y)^*$

by (*fact star-subdist-var-3*)

thus $(x^* \cdot y^*)^* \leq (x + y)^*$

by (*simp add: prod-star-closure star-inductl-star*)

qed

lemma *star-sum-var [simp]*: $(x^* + y^*)^* = (x + y)^*$

by *simp*

lemma *star-denest-var [simp]*: $x^* \cdot (y \cdot x^*)^* = (x + y)^*$

proof (*rule antisym*)

have $1 \leq x^* \cdot (y \cdot x^*)^*$

by (*metis mult-isol-var mult-oner star-ref*)

moreover have $x \cdot x^* \cdot (y \cdot x^*)^* \leq x^* \cdot (y \cdot x^*)^*$

by (*simp add: mult-isol*)

moreover have $y \cdot x^* \cdot (y \cdot x^*)^* \leq x^* \cdot (y \cdot x^*)^*$

by (*metis mult-isol-var mult-onel star-1l star-ref*)

ultimately have $1 + (x + y) \cdot x^* \cdot (y \cdot x^*)^* \leq x^* \cdot (y \cdot x^*)^*$

by *auto*

thus $(x + y)^* \leq x^* \cdot (y \cdot x^*)^*$

by (*metis mult.assoc mult-oner star-inductl*)

have $(y \cdot x^*)^* \leq (y^* \cdot x^*)^*$

by (*simp add: mult-isol-var star-iso*)

hence $(y \cdot x^*)^* \leq (x + y)^*$

by (*metis add commute star-denest*)

moreover have $x^* \leq (x + y)^*$

by (*fact star-subdist*)

ultimately show $x^* \cdot (y \cdot x^*)^* \leq (x + y)^*$

using *prod-star-closure* **by** *blast*

qed

lemma *star-denest-var-2* [*simp*]: $x^* \cdot (y \cdot x^*)^* = (x^* \cdot y^*)^*$
by *simp*

lemma *star-denest-var-3* [*simp*]: $x^* \cdot (y^* \cdot x^*)^* = (x^* \cdot y^*)^*$
by *simp*

lemma *star-denest-var-4* [*ac-simps*]: $(y^* \cdot x^*)^* = (x^* \cdot y^*)^*$
by (*metis add-comm star-denest*)

lemma *star-denest-var-5* [*ac-simps*]: $x^* \cdot (y \cdot x^*)^* = y^* \cdot (x \cdot y^*)^*$
by (*simp add: star-denest-var-4*)

lemma $x^* \cdot (y \cdot x^*)^* = (x^* \cdot y)^* \cdot x^*$

oops

lemma *star-denest-var-6* [*simp*]: $x^* \cdot y^* \cdot (x + y)^* = (x + y)^*$
using *mult-assoc* **by** *simp*

lemma *star-denest-var-7* [*simp*]: $(x + y)^* \cdot x^* \cdot y^* = (x + y)^*$
proof (*rule antisym*)

have $(x + y)^* \cdot x^* \cdot y^* \leq (x + y)^* \cdot (x^* \cdot y^*)^*$

by (*simp add: mult-assoc*)

thus $(x + y)^* \cdot x^* \cdot y^* \leq (x + y)^*$

by *simp*

have $1 \leq (x + y)^* \cdot x^* \cdot y^*$

by (*metis dual-order.trans mult-1-left mult-isor star-ref*)

moreover have $(x + y) \cdot (x + y)^* \cdot x^* \cdot y^* \leq (x + y)^* \cdot x^* \cdot y^*$

using *mult-isor star-11* **by** *presburger*

ultimately have $1 + (x + y) \cdot (x + y)^* \cdot x^* \cdot y^* \leq (x + y)^* \cdot x^* \cdot y^*$

by *simp*

thus $(x + y)^* \leq (x + y)^* \cdot x^* \cdot y^*$

by (*metis mult.assoc star-inductl-one*)

qed

lemma *star-denest-var-8* [*simp*]: $x^* \cdot y^* \cdot (x^* \cdot y^*)^* = (x^* \cdot y^*)^*$
by (*simp add: mult-assoc*)

lemma *star-denest-var-9* [*simp*]: $(x^* \cdot y^*)^* \cdot x^* \cdot y^* = (x^* \cdot y^*)^*$
using *star-denest-var-7* **by** *simp*

The following statements are well known from term rewriting. They are all variants of the Church-Rosser theorem in Kleene algebra [?]. But first we prove a law relating two confluence properties.

lemma *confluence-var* [*iff*]: $y \cdot x^* \leq x^* \cdot y^* \iff y^* \cdot x^* \leq x^* \cdot y^*$

proof

assume $y \cdot x^* \leq x^* \cdot y^*$

thus $y^* \cdot x^* \leq x^* \cdot y^*$

```

    using star-sim1 by fastforce
next
  assume  $y^* \cdot x^* \leq x^* \cdot y^*$ 
  thus  $y \cdot x^* \leq x^* \cdot y^*$ 
    by (meson mult-isol order-trans star-ext)
qed

```

```

lemma church-rosser [intro]:  $y^* \cdot x^* \leq x^* \cdot y^* \implies (x + y)^* = x^* \cdot y^*$ 
proof (rule antisym)
  assume  $y^* \cdot x^* \leq x^* \cdot y^*$ 
  hence  $x^* \cdot y^* \cdot (x^* \cdot y^*) \leq x^* \cdot x^* \cdot y^* \cdot y^*$ 
    by (metis mult-isol mult-isol mult.assoc)
  hence  $x^* \cdot y^* \cdot (x^* \cdot y^*) \leq x^* \cdot y^*$ 
    by (simp add: mult-assoc)
  moreover have  $1 \leq x^* \cdot y^*$ 
    by (metis dual-order.trans mult-1-right mult-isol star-ref)
  ultimately have  $1 + x^* \cdot y^* \cdot (x^* \cdot y^*) \leq x^* \cdot y^*$ 
    by simp
  hence  $(x^* \cdot y^*)^* \leq x^* \cdot y^*$ 
    by (simp add: star-inductl-one)
  thus  $(x + y)^* \leq x^* \cdot y^*$ 
    by simp
  thus  $x^* \cdot y^* \leq (x + y)^*$ 
    by simp
qed

```

```

lemma church-rosser-var:  $y \cdot x^* \leq x^* \cdot y^* \implies (x + y)^* = x^* \cdot y^*$ 
  by fastforce

```

```

lemma church-rosser-to-confluence:  $(x + y)^* \leq x^* \cdot y^* \implies y^* \cdot x^* \leq x^* \cdot y^*$ 
  by (metis add-comm eq-iff star-subdist-var-3)

```

```

lemma church-rosser-equiv:  $y^* \cdot x^* \leq x^* \cdot y^* \longleftrightarrow (x + y)^* = x^* \cdot y^*$ 
  using church-rosser-to-confluence eq-iff by blast

```

```

lemma confluence-to-local-confluence:  $y^* \cdot x^* \leq x^* \cdot y^* \implies y \cdot x \leq x^* \cdot y^*$ 
  by (meson mult-isol-var order-trans star-ext)

```

```

lemma  $y \cdot x \leq x^* \cdot y^* \implies y^* \cdot x^* \leq x^* \cdot y^*$ 

```

oops

```

lemma  $y \cdot x \leq x^* \cdot y^* \implies (x + y)^* \leq x^* \cdot y^*$ 
  oops

```

More variations could easily be proved. The last counterexample shows that Newman's lemma needs a wellfoundedness assumption. This is well known.

The next lemmas relate the reflexive transitive closure and the transitive

closure.

lemma *sup-id-star1*: $1 \leq x \implies x \cdot x^* = x^*$

proof –

assume $1 \leq x$

hence $x^* \leq x \cdot x^*$

using *mult-isol* **by** *fastforce*

thus $x \cdot x^* = x^*$

by (*simp add: eq-iff*)

qed

lemma *sup-id-star2*: $1 \leq x \implies x^* \cdot x = x^*$

by (*metis order.antisym mult-isol mult-oner star-1r*)

lemma $1 + x^* \cdot x = x^*$

oops

lemma $(x \cdot y)^* \cdot x = x \cdot (y \cdot x)^*$

oops

lemma $x \cdot x = x \implies x^* = 1 + x$

oops

end

0.11.3 Left Kleene Algebras

class *left-kleene-algebra* = *left-pre-kleene-algebra* + *dioide-one*

begin

In left Kleene algebras the non-fact $z + y \cdot x \leq y \implies z \cdot x^* \leq y$ is a good challenge for counterexample generators. A model of left Kleene algebras in which the right star induction law does not hold has been given by Kozen [?].

We now show that the right unfold law becomes an equality.

lemma *star-unfoldr-eq* [*simp*]: $1 + x^* \cdot x = x^*$

proof (*rule antisym*)

show $1 + x^* \cdot x \leq x^*$

by (*fact star-unfoldr*)

have $1 + x \cdot (1 + x^* \cdot x) = 1 + (1 + x \cdot x^*) \cdot x$

using *distrib-left distrib-right mult-1-left mult-1-right mult-assoc* **by** *presburger*

also have $\dots = 1 + x^* \cdot x$

by *simp*

finally show $x^* \leq 1 + x^* \cdot x$

by (*simp add: star-inductl-one*)

qed

The following more complex unfold law has been used as an axiom, called *prodstar*, by Conway [?].

lemma *star-prod-unfold* [*simp*]: $1 + x \cdot (y \cdot x)^* \cdot y = (x \cdot y)^*$

proof (*rule antisym*)

have $(x \cdot y)^* = 1 + (x \cdot y)^* \cdot x \cdot y$

by (*simp add: mult-assoc*)

thus $(x \cdot y)^* \leq 1 + x \cdot (y \cdot x)^* \cdot y$

by (*metis join.sup-mono mult-isol order-refl star-slide1*)

have $1 + x \cdot (y \cdot x)^* \cdot y \leq 1 + x \cdot y \cdot (x \cdot y)^*$

by (*metis join.sup-mono eq-refl mult.assoc mult-isol star-slide1*)

thus $1 + x \cdot (y \cdot x)^* \cdot y \leq (x \cdot y)^*$

by *simp*

qed

The slide laws, which have previously been inequalities, now become equations.

lemma *star-slide* [*ac-simps*]: $(x \cdot y)^* \cdot x = x \cdot (y \cdot x)^*$

proof –

have $x \cdot (y \cdot x)^* = x \cdot (1 + y \cdot (x \cdot y)^* \cdot x)$

by *simp*

also have $\dots = (1 + x \cdot y \cdot (x \cdot y)^*) \cdot x$

by (*simp add: distrib-left mult-assoc*)

finally show *?thesis*

by *simp*

qed

lemma *star-slide-var* [*ac-simps*]: $x^* \cdot x = x \cdot x^*$

by (*metis mult-onel mult-oner star-slide*)

lemma *star-sum-unfold-var* [*simp*]: $1 + x^* \cdot (x + y)^* \cdot y^* = (x + y)^*$

by (*metis star-denest star-denest-var-3 star-denest-var-4 star-plus-one star-slide*)

The following law shows how starred sums can be unfolded.

lemma *star-sum-unfold* [*simp*]: $x^* + x^* \cdot y \cdot (x + y)^* = (x + y)^*$

proof –

have $(x + y)^* = x^* \cdot (y \cdot x^*)^*$

by *simp*

also have $\dots = x^* \cdot (1 + y \cdot x^* \cdot (y \cdot x^*)^*)$

by *simp*

also have $\dots = x^* \cdot (1 + y \cdot (x + y)^*)$

by (*simp add: mult.assoc*)

finally show *?thesis*

by (*simp add: distrib-left mult-assoc*)

qed

The following property appears in process algebra.

lemma *troeger*: $(x + y)^* \cdot z = x^* \cdot (y \cdot (x + y)^* \cdot z + z)$

proof –


```

have  $(x + y)^* \cdot z = x^* \cdot z + x^* \cdot y \cdot (x + y)^* \cdot z$ 
  by (metis (full-types) distrib-right star-sum-unfold)
thus ?thesis
  by (simp add: add-commute distrib-left mult-assoc)
qed

```

The following properties are related to a property from propositional dynamic logic which has been attributed to Albert Meyer [?]. Here we prove it as a theorem of Kleene algebra.

lemma *star-square*: $(x \cdot x)^* \leq x^*$

proof –

```

have  $x \cdot x \cdot x^* \leq x^*$ 
  by (simp add: prod-star-closure)
thus ?thesis
  by (simp add: star-inductl-star)
qed

```

lemma *meyer-1* [simp]: $(1 + x) \cdot (x \cdot x)^* = x^*$

proof (*rule antisym*)

```

have  $x \cdot (1 + x) \cdot (x \cdot x)^* = x \cdot (x \cdot x)^* + x \cdot x \cdot (x \cdot x)^*$ 
  by (simp add: distrib-left)
also have  $\dots \leq x \cdot (x \cdot x)^* + (x \cdot x)^*$ 
  using join.sup-mono star-1l by blast
finally have  $x \cdot (1 + x) \cdot (x \cdot x)^* \leq (1 + x) \cdot (x \cdot x)^*$ 
  by (simp add: join.sup-commute)
moreover have  $1 \leq (1 + x) \cdot (x \cdot x)^*$ 
  using join.sup.cobounded11 by auto
ultimately have  $1 + x \cdot (1 + x) \cdot (x \cdot x)^* \leq (1 + x) \cdot (x \cdot x)^*$ 
  by auto
thus  $x^* \leq (1 + x) \cdot (x \cdot x)^*$ 
  by (simp add: star-inductl-one mult-assoc)
show  $(1 + x) \cdot (x \cdot x)^* \leq x^*$ 
  by (simp add: prod-star-closure star-square)
qed

```

The following lemma says that transitive elements are equal to their transitive closure.

lemma *tc*: $x \cdot x \leq x \implies x^* \cdot x = x$

proof –

```

assume  $x \cdot x \leq x$ 
hence  $x + x \cdot x \leq x$ 
  by simp
hence  $x^* \cdot x \leq x$ 
  by (fact star-inductl)
thus  $x^* \cdot x = x$ 
  by (metis mult-isol mult-oner star-ref star-slide-var eq-iff)
qed

```

lemma *tc-eq*: $x \cdot x = x \implies x^* \cdot x = x$

by (*auto intro: tc*)

The next fact has been used by Boffa [?] to axiomatise the equational theory of regular expressions.

lemma *boffa-var*: $x \cdot x \leq x \implies x^* = 1 + x$

proof –

assume $x \cdot x \leq x$

moreover have $x^* = 1 + x^* \cdot x$

by *simp*

ultimately show $x^* = 1 + x$

by (*simp add: tc*)

qed

lemma *boffa*: $x \cdot x = x \implies x^* = 1 + x$

by (*auto intro: boffa-var*)

end

0.11.4 Left Kleene Algebras with Zero

There are applications where only a left zero is assumed, for instance in the context of total correctness and for demonic refinement algebras [?].

class *left-kleene-algebra-zero* = *left-kleene-algebra* + *dioid-one-zero*

begin

sublocale *conway*: *near-conway-base-zero* *star*

by *standard* (*simp-all add: local.star-slide*)

lemma *star-zero* [*simp*]: $0^* = 1$

by (*rule local.conway.zero-dagger*)

In principle, 1 could therefore be defined from 0 in this setting.

end

class *left-kleene-algebra-zero* = *left-kleene-algebra-zero* + *dioid-one-zero*

0.11.5 Pre-Kleene Algebras

Pre-Kleene algebras are essentially probabilistic Kleene algebras [?]. They have a weaker right star unfold axiom. We are still looking for theorems that could be proved in this setting.

class *pre-kleene-algebra* = *left-pre-kleene-algebra* +

assumes *weak-star-unfoldr*: $z + y \cdot (x + 1) \leq y \implies z \cdot x^* \leq y$

0.11.6 Kleene Algebras

class *kleene-algebra-zero1* = *left-kleene-algebra-zero1* +
assumes *star-inductr*: $z + y \cdot x \leq y \implies z \cdot x^* \leq y$

begin

lemma *star-sim2*: $z \cdot x \leq y \cdot z \implies z \cdot x^* \leq y^* \cdot z$

proof –

assume $z \cdot x \leq y \cdot z$

hence $y^* \cdot z \cdot x \leq y^* \cdot y \cdot z$

using *mult-isol mult-assoc* **by** *auto*

also have $\dots \leq y^* \cdot z$

by (*simp add: mult-isol*)

finally have $y^* \cdot z \cdot x \leq y^* \cdot z$

by *simp*

moreover have $z \leq y^* \cdot z$

using *mult-isol star-ref* **by** *fastforce*

ultimately have $z + y^* \cdot z \cdot x \leq y^* \cdot z$

by *simp*

thus $z \cdot x^* \leq y^* \cdot z$

by (*simp add: star-inductr*)

qed

sublocale *conway*: *pre-conway star*

by *standard* (*simp add: star-sim2*)

lemma *star-inductr-var*: $y \cdot x \leq y \implies y \cdot x^* \leq y$

by (*simp add: star-inductr*)

lemma *star-inductr-var-equiv*: $y \cdot x \leq y \iff y \cdot x^* \leq y$

by (*meson order-trans mult-isol star-ext star-inductr-var*)

lemma *star-sim3*: $z \cdot x = y \cdot z \implies z \cdot x^* = y^* \cdot z$

by (*simp add: eq-iff star-sim1 star-sim2*)

lemma *star-sim4*: $x \cdot y \leq y \cdot x \implies x^* \cdot y^* \leq y^* \cdot x^*$

by (*auto intro: star-sim1 star-sim2*)

lemma *star-inductr-eq*: $z + y \cdot x = y \implies z \cdot x^* \leq y$

by (*auto intro: star-inductr*)

lemma *star-inductr-var-eq*: $y \cdot x = y \implies y \cdot x^* \leq y$

by (*auto intro: star-inductr-var*)

lemma *star-inductr-var-eq2*: $y \cdot x = y \implies y \cdot x^* = y$

by (*metis mult-one1 star-one star-sim3*)

lemma *bubble-sort*: $y \cdot x \leq x \cdot y \implies (x + y)^* = x^* \cdot y^*$

by (*fastforce intro: star-sim4*)

```

lemma independence1:  $x \cdot y = 0 \implies x^* \cdot y = y$ 
proof –
  assume  $x \cdot y = 0$ 
  moreover have  $x^* \cdot y = y + x^* \cdot x \cdot y$ 
    by (metis distrib-right mult-onel star-unfoldr-eq)
  ultimately show  $x^* \cdot y = y$ 
    by (metis add-0-left add.commute join.sup-ge1 eq-iff star-inductl-eq)
qed

lemma independence2:  $x \cdot y = 0 \implies x \cdot y^* = x$ 
  by (metis annil mult-onel star-sim3 star-zero)

lemma lazycomm-var:  $y \cdot x \leq x \cdot (x + y)^* + y \longleftrightarrow y \cdot x^* \leq x \cdot (x + y)^* + y$ 
proof
  let  $?t = x \cdot (x + y)^*$ 
  assume hyp:  $y \cdot x \leq ?t + y$ 
  have  $(?t + y) \cdot x = ?t \cdot x + y \cdot x$ 
    by (fact distrib-right)
  also have  $\dots \leq ?t \cdot x + ?t + y$ 
    using hyp join.sup.coboundedI2 join.sup-assoc by auto
  also have  $\dots \leq ?t + y$ 
    using eq-refl join.sup-least join.sup-mono mult-isol prod-star-closure star-subdist-var-1
  mult-assoc by presburger
  finally have  $y + (?t + y) \cdot x \leq ?t + y$ 
    by simp
  thus  $y \cdot x^* \leq x \cdot (x + y)^* + y$ 
    by (fact star-inductr)
next
  assume  $y \cdot x^* \leq x \cdot (x + y)^* + y$ 
  thus  $y \cdot x \leq x \cdot (x + y)^* + y$ 
    using dual-order.trans mult-isol star-ext by blast
qed

lemma arden-var:  $(\forall y \ v. y \leq x \cdot y + v \longrightarrow y \leq x^* \cdot v) \implies z = x \cdot z + w \implies z = x^* \cdot w$ 
  by (auto simp: add-comm eq-iff star-inductl-eq)

lemma  $(\forall x \ y. y \leq x \cdot y \longrightarrow y = 0) \implies y \leq x \cdot y + z \implies y \leq x^* \cdot z$ 
  by (metis eq-refl mult-onel)

end

```

Finally, here come the Kleene algebras à la Kozen [?]. We only prove quasi-identities in this section. Since left Kleene algebras are complete with respect to the equational theory of regular expressions and regular languages, all identities hold already without the right star induction axiom.

```

class kleene-algebra = left-kleene-algebra-zero +
  assumes star-inductr':  $z + y \cdot x \leq y \implies z \cdot x^* \leq y$ 

```

begin

```
subclass kleene-algebra-zero
  by standard (simp add: star-inductr')

sublocale conway-zero: conway star ..
```

The next lemma shows that opposites of Kleene algebras (i.e., Kleene algebras with the order of multiplication swapped) are again Kleene algebras.

lemma *dual-kleene-algebra*:

```
class.kleene-algebra (+) ( $\odot$ ) 1 0 ( $\leq$ ) ( $<$ ) star
proof
  fix x y z :: 'a
  show  $(x \odot y) \odot z = x \odot (y \odot z)$ 
    by (metis mult.assoc opp-mult-def)
  show  $(x + y) \odot z = x \odot z + y \odot z$ 
    by (metis opp-mult-def distrib-left)
  show  $1 \odot x = x$ 
    by (metis mult-oner opp-mult-def)
  show  $x \odot 1 = x$ 
    by (metis mult-onel opp-mult-def)
  show  $0 + x = x$ 
    by (fact add-zero)
  show  $0 \odot x = 0$ 
    by (metis annir opp-mult-def)
  show  $x \odot 0 = 0$ 
    by (metis annil opp-mult-def)
  show  $x + x = x$ 
    by (fact add-idem)
  show  $x \odot (y + z) = x \odot y + x \odot z$ 
    by (metis distrib-right opp-mult-def)
  show  $z \odot x \leq z \odot (x + y)$ 
    by (metis mult-isor opp-mult-def order-prop)
  show  $1 + x \odot x^* \leq x^*$ 
    by (metis opp-mult-def order-refl star-slide-var star-unfoldl-eq)
  show  $z + x \odot y \leq y \implies x^* \odot z \leq y$ 
    by (metis opp-mult-def star-inductr)
  show  $z + y \odot x \leq y \implies z \odot x^* \leq y$ 
    by (metis opp-mult-def star-inductl)
qed
```

end

We finish with some properties on (multiplicatively) commutative Kleene algebras. A chapter in Conway's book [?] is devoted to this topic.

```
class commutative-kleene-algebra = kleene-algebra +
  assumes mult-comm [ac-simps]:  $x \cdot y = y \cdot x$ 
```

begin

```

lemma conway-c3 [simp]:  $(x + y)^* = x^* \cdot y^*$ 
  using church-rosser mult-comm by auto

lemma conway-c4:  $(x^* \cdot y)^* = 1 + x^* \cdot y^* \cdot y$ 
  by (metis conway-c3 star-denest-var star-prod-unfold)

lemma cka-1:  $(x \cdot y)^* \leq x^* \cdot y^*$ 
  by (metis conway-c3 star-invol star-iso star-subdist-var-2)

lemma cka-2 [simp]:  $x^* \cdot (x^* \cdot y)^* = x^* \cdot y^*$ 
  by (metis conway-c3 mult-comm star-denest-var)

lemma conway-c4-var [simp]:  $(x^* \cdot y^*)^* = x^* \cdot y^*$ 
  by (metis conway-c3 star-invol)

lemma conway-c2-var:  $(x \cdot y)^* \cdot x \cdot y \cdot y^* \leq (x \cdot y)^* \cdot y^*$ 
  by (metis mult-isor star-1r mult-assoc)

lemma conway-c2 [simp]:  $(x \cdot y)^* \cdot (x^* + y^*) = x^* \cdot y^*$ 
proof (rule antisym)
  show  $(x \cdot y)^* \cdot (x^* + y^*) \leq x^* \cdot y^*$ 
    by (metis cka-1 conway-c3 prod-star-closure star-ext star-sum-var)
  have  $x \cdot (x \cdot y)^* \cdot (x^* + y^*) = x \cdot (x \cdot y)^* \cdot (x^* + 1 + y \cdot y^*)$ 
    by (simp add: add-assoc)
  also have  $\dots = x \cdot (x \cdot y)^* \cdot (x^* + y \cdot y^*)$ 
    by (simp add: add-commute)
  also have  $\dots = (x \cdot y)^* \cdot (x \cdot x^*) + (x \cdot y)^* \cdot x \cdot y \cdot y^*$ 
    using distrib-left mult-comm mult-assoc by force
  also have  $\dots \leq (x \cdot y)^* \cdot x^* + (x \cdot y)^* \cdot x \cdot y \cdot y^*$ 
    using add-iso mult-isol by force
  also have  $\dots \leq (x \cdot y)^* \cdot x^* + (x \cdot y)^* \cdot y^*$ 
    using conway-c2-var join.sup-mono by blast
  also have  $\dots = (x \cdot y)^* \cdot (x^* + y^*)$ 
    by (simp add: distrib-left)
  finally have  $x \cdot (x \cdot y)^* \cdot (x^* + y^*) \leq (x \cdot y)^* \cdot (x^* + y^*)$  .
  moreover have  $y^* \leq (x \cdot y)^* \cdot (x^* + y^*)$ 
    by (metis dual-order.trans join.sup-ge2 mult-1-left mult-isor star-ref)
  ultimately have  $y^* + x \cdot (x \cdot y)^* \cdot (x^* + y^*) \leq (x \cdot y)^* \cdot (x^* + y^*)$ 
    by simp
  thus  $x^* \cdot y^* \leq (x \cdot y)^* \cdot (x^* + y^*)$ 
    by (simp add: mult.assoc star-inductl)
qed

end

end

```

0.12 Models of Dioids

```
theory Dioid-Models
imports Kleene-Algebra.Dioid HOL.Real
begin
```

In this section we consider some well known models of dioids. These so far include the powerset dioid over a monoid, languages, binary relations, sets of traces, sets paths (in a graph), as well as the min-plus and the max-plus semirings. Most of these models are taken from an article about Kleene algebras with domain [?].

The advantage of formally linking these models with the abstract axiomatisations of dioids is that all abstract theorems are automatically available in all models. It therefore makes sense to establish models for the strongest possible axiomatisations (whereas theorems should be proved for the weakest ones).

0.12.1 The Powerset Dioid over a Monoid

We assume a multiplicative monoid and define the usual complex product on sets of elements. We formalise the well known result that this lifting induces a dioid.

0.12.2 Language Dioids

Language dioids arise as special cases of the monoidal lifting because sets of words form free monoids. Moreover, monoids of words are isomorphic to monoids of lists under append.

To show that languages form dioids it therefore suffices to show that sets of lists closed under append and multiplication with the empty word form a (multiplicative) monoid. Isabelle then does the rest of the work automatically. Infix @ denotes word concatenation.

```
instantiation list :: (type) monoid-mult
begin
```

```
definition times-list-def:
   $xs * ys \equiv xs @ ys$ 
```

```
definition one-list-def:
   $1 \equiv []$ 
```

```
instance proof
fix xs ys zs :: 'a list'
show  $xs * ys * zs = xs * (ys * zs)$ 
by (simp add: times-list-def)
```

```

show 1 * xs = xs
  by (simp add: one-list-def times-list-def)
show xs * 1 = xs
  by (simp add: one-list-def times-list-def)
qed

end

```

0.12.3 Relation Dioids

We now show that binary relations under union, relational composition, the identity relation, the empty relation and set inclusion form dioids. Due to the well developed relation library of Isabelle this is entirely trivial.

```

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

```

```

interpretation rel-monoid: monoid-mult Id ( $\circ$ ) ..

```

0.12.4 Trace Dioids

Traces have been considered, for instance, by Kozen [?] in the context of Kleene algebras with tests. Intuitively, a trace is an execution sequence of a labelled transition system from some state to some other state, in which state labels and action labels alternate, and which begin and end with a state label.

Traces generalise words: words can be obtained from traces by forgetting state labels. Similarly, sets of traces generalise languages.

In this section we show that sets of traces under union, an appropriately defined notion of complex product, the set of all traces of length zero, the empty set of traces and set inclusion form a dioid.

We first define the notion of trace and the product of traces, which has been called *fusion product* by Kozen.

```

type-synonym ('p, 'a) trace = 'p  $\times$  ('a  $\times$  'p) list

```

```

definition first :: ('p, 'a) trace  $\Rightarrow$  'p where
  first = fst

```

```

lemma first-conv [simp]: first (p, xs) = p
  by (unfold first-def, simp)

```

```

fun last :: ('p, 'a) trace  $\Rightarrow$  'p where
  last (p, []) = p
| last (_, xs) = snd (List.last xs)

```

```

lemma last-append [simp]: last (p, xs @ ys) = last (last (p, xs), ys)
proof (cases xs)

```



```

show  $xs = [] \implies \text{last } (p, xs @ ys) = \text{last } (\text{last } (p, xs), ys)$ 
  by simp
show  $\bigwedge a \text{ list. } xs = a \# list \implies$ 
   $\text{last } (p, xs @ ys) = \text{last } (\text{last } (p, xs), ys)$ 
proof (cases ys)
  show  $\bigwedge a \text{ list. } [xs = a \# list; ys = []]$ 
     $\implies \text{last } (p, xs @ ys) = \text{last } (\text{last } (p, xs), ys)$ 
    by simp
  show  $\bigwedge a \text{ list } aa \text{ lista. } [xs = a \# list; ys = aa \# lista]$ 
     $\implies \text{last } (p, xs @ ys) = \text{last } (\text{last } (p, xs), ys)$ 
    by simp
qed
qed

```

The fusion product is a partial operation. It is undefined if the last element of the first trace and the first element of the second trace are different. If these elements are the same, then the fusion product removes the first element from the second trace and appends the resulting object to the first trace.

definition $t\text{-fusion} :: ('p, 'a) \text{ trace} \Rightarrow ('p, 'a) \text{ trace} \Rightarrow ('p, 'a) \text{ trace}$ **where**
 $t\text{-fusion } x \ y \equiv \text{if last } x = \text{first } y \text{ then } (\text{fst } x, \text{snd } x @ \text{snd } y) \text{ else undefined}$

We now show that the first element and the last element of a trace are a left and right unit for that trace and prove some other auxiliary lemmas.

lemma $t\text{-fusion-leftneutral}$ [*simp*]: $t\text{-fusion } (\text{first } x, []) \ x = x$
by (*cases x, simp add: t-fusion-def*)

lemma $t\text{-fusion-rightneutral}$ [*simp*]: $t\text{-fusion } x \ (\text{last } x, []) = x$
by (*simp add: t-fusion-def*)

lemma $\text{first-}t\text{-fusion}$ [*simp*]: $\text{last } x = \text{first } y \implies \text{first } (t\text{-fusion } x \ y) = \text{first } x$
by (*simp add: first-def t-fusion-def*)

lemma $\text{last-}t\text{-fusion}$ [*simp*]: $\text{last } x = \text{first } y \implies \text{last } (t\text{-fusion } x \ y) = \text{last } y$
by (*simp add: first-def t-fusion-def*)

Next we show that fusion of traces is associative.

lemma $t\text{-fusion-assoc}$ [*simp*]:
 $[\text{last } x = \text{first } y; \text{last } y = \text{first } z] \implies t\text{-fusion } x \ (t\text{-fusion } y \ z) = t\text{-fusion } (t\text{-fusion } x \ y) \ z$
by (*cases x, cases y, cases z, simp add: t-fusion-def*)

0.12.5 Sets of Traces

We now lift the fusion product to a complex product on sets of traces. This operation is total.

no-notation

times (**infixl** · 70)

definition $t\text{-prod} :: ('p, 'a) \text{ trace set} \Rightarrow ('p, 'a) \text{ trace set} \Rightarrow ('p, 'a) \text{ trace set}$
(infixl · 70)

where $X \cdot Y = \{t\text{-fusion } u \ v \mid u \ v. \ u \in X \wedge v \in Y \wedge \text{last } u = \text{first } v\}$

Next we define the empty set of traces and the set of traces of length zero as the multiplicative unit of the trace dioid.

definition $t\text{-zero} :: ('p, 'a) \text{ trace set}$ **where**
 $t\text{-zero} \equiv \{\}$

definition $t\text{-one} :: ('p, 'a) \text{ trace set}$ **where**
 $t\text{-one} \equiv \bigcup p. \{(p, [])\}$

We now provide elimination rules for trace products.

lemma $t\text{-prod-iff}$:
 $w \in X \cdot Y \iff (\exists u \ v. \ w = t\text{-fusion } u \ v \wedge u \in X \wedge v \in Y \wedge \text{last } u = \text{first } v)$
by (*unfold t-prod-def*) *auto*

lemma $t\text{-prod-intro}$ [*simp, intro*]:
 $\llbracket u \in X; v \in Y; \text{last } u = \text{first } v \rrbracket \implies t\text{-fusion } u \ v \in X \cdot Y$
by (*meson t-prod-iff*)

lemma $t\text{-prod-elim}$ [*elim*]:
 $w \in X \cdot Y \implies \exists u \ v. \ w = t\text{-fusion } u \ v \wedge u \in X \wedge v \in Y \wedge \text{last } u = \text{first } v$
by (*meson t-prod-iff*)

Finally we prove the interpretation statement that sets of traces under union and the complex product based on trace fusion together with the empty set of traces and the set of traces of length one forms a dioid.

interpretation trace-diod : $\text{diod-one-zero } (\cup) \ t\text{-prod } t\text{-one } t\text{-zero } (\subseteq) (\subset)$
apply *unfold-locales*
apply (*auto simp add: t-prod-def t-one-def t-zero-def t-fusion-def*)
apply (*metis last-append*)
apply (*metis last-append append-assoc*)
done

no-notation
 $t\text{-prod}$ (**infixl** · 70)

0.12.6 The Path Diod

The next model we consider are sets of paths in a graph. We consider two variants, one that contains the empty path and one that doesn't. The former leads to more difficult proofs and a more involved specification of the complex product. We start with paths that include the empty path. In this setting, a path is a list of nodes.

0.12.7 Path Models with the Empty Path

type-synonym $'a \text{ path} = 'a \text{ list}$

Path fusion is defined similarly to trace fusion. Mathematically it should be a partial operation. The fusion of two empty paths yields the empty path; the fusion between a non-empty path and an empty one is undefined; the fusion of two non-empty paths appends the tail of the second path to the first one.

We need to use a total alternative and make sure that undefined paths do not contribute to the complex product.

fun $p\text{-fusion} :: 'a \text{ path} \Rightarrow 'a \text{ path} \Rightarrow 'a \text{ path}$ **where**

$p\text{-fusion } [] - = []$
 $| p\text{-fusion } - [] = []$
 $| p\text{-fusion } ps (q \# qs) = ps @ qs$

lemma $p\text{-fusion-assoc}$:

$p\text{-fusion } ps (p\text{-fusion } qs rs) = p\text{-fusion } (p\text{-fusion } ps qs) rs$
proof (*induct rs*)
case Nil show $?case$
by (*metis p-fusion.elims p-fusion.simps(2)*)
case Cons show $?case$
proof (*induct qs*)
case Nil show $?case$
by (*metis neq-Nil-conv p-fusion.simps(1) p-fusion.simps(2)*)
case Cons show $?case$
proof –
have $\forall ps. ([] = ps \vee \text{hd } ps \# \text{tl } ps = ps) \wedge ((\forall q qs. q \# qs \neq ps) \vee [] \neq ps)$
using *list.collapse* **by** *fastforce*
moreover hence $\forall ps q qs. p\text{-fusion } ps (q \# qs) = ps @ qs \vee [] = ps$
by (*metis p-fusion.simps(3)*)
ultimately show $?thesis$
by (*metis (no-types) Cons-eq-appendI append-eq-appendI p-fusion.simps(1) p-fusion.simps(3)*)
qed
qed
qed

This lemma overapproximates the real situation, but it holds in all cases where path fusion should be defined.

lemma $p\text{-fusion-last}$:

assumes $\text{List.last } ps = \text{hd } qs$
and $ps \neq []$
and $qs \neq []$
shows $\text{List.last } (p\text{-fusion } ps qs) = \text{List.last } qs$
by (*metis (hide-lams, no-types) List.last.simps List.last-append append-Nil2 assms list.sel(1) neq-Nil-conv p-fusion.simps(3)*)

lemma *p-fusion-hd*: $\llbracket ps \neq []; qs \neq [] \rrbracket \implies hd (p\text{-fusion } ps \ qs) = hd \ ps$
by (*metis list.exhaust p-fusion.simps(3) append-Cons list.sel(1)*)

lemma *nonempty-p-fusion*: $\llbracket ps \neq []; qs \neq [] \rrbracket \implies p\text{-fusion } ps \ qs \neq []$
by (*metis list.exhaust append-Cons p-fusion.simps(3) list.simps(2)*)

We now define a condition that filters out undefined paths in the complex product.

abbreviation *p-filter* :: 'a path \Rightarrow 'a path \Rightarrow bool **where**
p-filter *ps qs* $\equiv ((ps = [] \wedge qs = []) \vee (ps \neq [] \wedge qs \neq [] \wedge (List.last \ ps) = hd \ qs))$

no-notation
times (**infixl** \cdot 70)

definition *p-prod* :: 'a path set \Rightarrow 'a path set \Rightarrow 'a path set (**infixl** \cdot 70)
where $X \cdot Y = \{rs \mid \exists ps \in X. \exists qs \in Y. rs = p\text{-fusion } ps \ qs \wedge p\text{-filter } ps \ qs\}$

lemma *p-prod-iff*:
 $ps \in X \cdot Y \iff (\exists qs \ rs. ps = p\text{-fusion } qs \ rs \wedge qs \in X \wedge rs \in Y \wedge p\text{-filter } qs \ rs)$
by (*unfold p-prod-def*) *auto*

Due to the complexity of the filter condition, proving properties of complex products can be tedious.

lemma *p-prod-assoc*: $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
proof (*rule set-eqI*)
fix *ps*
show $ps \in (X \cdot Y) \cdot Z \iff ps \in X \cdot (Y \cdot Z)$
proof (*cases ps*)
case *Nil* **thus** ?thesis
by *auto* (*metis nonempty-p-fusion p-prod-iff*) +
next
case *Cons* **thus** ?thesis
by (*auto simp add: p-prod-iff*) (*metis* (*hide-lams, mono-tags*) *nonempty-p-fusion p-fusion-assoc p-fusion-hd p-fusion-last*) +
qed
qed

We now define the multiplicative unit of the path dioid as the set of all paths of length one, including the empty path, and show the unit laws with respect to the path product.

definition *p-one* :: 'a path set **where**
 $p\text{-one} \equiv \{p \mid \exists q::'a. p = [q]\} \cup \{[]\}$

lemma *p-prod-oneI* [*simp*]: $p\text{-one} \cdot X = X$
proof (*rule set-eqI*)
fix *ps*

```

show  $ps \in p\text{-one} \cdot X \longleftrightarrow ps \in X$ 
proof (cases  $ps$ )
  case  $Nil$  thus ?thesis
  by (auto simp add:  $p\text{-one-def}$   $p\text{-prod-def}$ ,  $metis$   $nonempty\text{-}p\text{-fusion}$   $not\text{-}Cons\text{-}self$ )
next
  case  $Cons$  thus ?thesis
  by (auto simp add:  $p\text{-one-def}$   $p\text{-prod-def}$ ,  $metis$   $append\text{-}Cons$   $append\text{-}Nil$ 
 $list.sel(1)$   $neq\text{-}Nil\text{-}conv$   $p\text{-fusion}.simps(3)$ ,  $metis$   $Cons\text{-}eq\text{-}appendI$   $list.sel(1)$   $last\text{-}ConsL$ 
 $list.simps(3)$   $p\text{-fusion}.simps(3)$   $self\text{-}append\text{-}conv2$ )
qed
qed

```

```

lemma  $p\text{-prod-oner}$  [simp]:  $X \cdot p\text{-one} = X$ 
proof (rule  $set\text{-}eqI$ )
  fix  $ps$ 
  show  $ps \in X \cdot p\text{-one} \longleftrightarrow ps \in X$ 
  proof (cases  $ps$ )
    case  $Nil$  thus ?thesis
    by (auto simp add:  $p\text{-one-def}$   $p\text{-prod-def}$ ,  $metis$   $nonempty\text{-}p\text{-fusion}$   $not\text{-}Cons\text{-}self2$ ,
 $metis$   $p\text{-fusion}.simps(1)$ )
  next
    case  $Cons$  thus ?thesis
    by (auto simp add:  $p\text{-one-def}$   $p\text{-prod-def}$ ,  $metis$   $append\text{-}Nil2$   $neq\text{-}Nil\text{-}conv$ 
 $p\text{-fusion}.simps(3)$ ,  $metis$   $list.sel(1)$   $list.simps(2)$   $p\text{-fusion}.simps(3)$   $self\text{-}append\text{-}conv$ )
  qed
qed

```

Next we show distributivity laws at the powerset level.

```

lemma  $p\text{-prod-distl}$ :  $X \cdot (Y \cup Z) = X \cdot Y \cup X \cdot Z$ 
proof (rule  $set\text{-}eqI$ )
  fix  $ps$ 
  show  $ps \in X \cdot (Y \cup Z) \longleftrightarrow ps \in X \cdot Y \cup X \cdot Z$ 
  by (cases  $ps$ ) (auto simp add:  $p\text{-prod-iff}$ )
qed

```

```

lemma  $p\text{-prod-distr}$ :  $(X \cup Y) \cdot Z = X \cdot Z \cup Y \cdot Z$ 
proof (rule  $set\text{-}eqI$ )
  fix  $ps$ 
  show  $ps \in (X \cup Y) \cdot Z \longleftrightarrow ps \in X \cdot Z \cup Y \cdot Z$ 
  by (cases  $ps$ ) (auto simp add:  $p\text{-prod-iff}$ )
qed

```

Finally we show that sets of paths under union, the complex product, the unit set and the empty set form a dioid.

```

interpretation  $path\text{-}dioid$ :  $dioid\text{-}one\text{-}zero$  ( $\cup$ ) ( $\cdot$ )  $p\text{-one}$   $\{\}$  ( $\subseteq$ ) ( $\subset$ )
proof
  fix  $x\ y\ z :: 'a\ path\ set$ 
  show  $x \cup y \cup z = x \cup (y \cup z)$ 

```

```

    by auto
  show  $x \cup y = y \cup x$ 
    by auto
  show  $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ 
    by (fact p-prod-assoc)
  show  $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$ 
    by (fact p-prod-distr)
  show  $p-one \cdot x = x$ 
    by (fact p-prod-one1)
  show  $x \cdot p-one = x$ 
    by (fact p-prod-one2)
  show  $\{\} \cup x = x$ 
    by auto
  show  $\{\} \cdot x = \{\}$ 
    by (metis all-not-in-conv p-prod-iff)
  show  $x \cdot \{\} = \{\}$ 
    by (metis all-not-in-conv p-prod-iff)
  show  $(x \subseteq y) = (x \cup y = y)$ 
    by auto
  show  $(x \subset y) = (x \subseteq y \wedge x \neq y)$ 
    by auto
  show  $x \cup x = x$ 
    by auto
  show  $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$ 
    by (fact p-prod-distl)
qed

```

no-notation

$p\text{-prod}$ (infixl 70)

0.12.8 Path Models without the Empty Path

We now build a model of paths that does not include the empty path and therefore leads to a simpler complex product.

datatype $'a$ ppath = Node $'a$ | Cons $'a$ $'a$ ppath

primrec $pp\text{-first} :: 'a \text{ ppath} \Rightarrow 'a$ **where**

$pp\text{-first} \text{ (Node } x) = x$
 $| pp\text{-first} \text{ (Cons } x \text{ -)} = x$

primrec $pp\text{-last} :: 'a \text{ ppath} \Rightarrow 'a$ **where**

$pp\text{-last} \text{ (Node } x) = x$
 $| pp\text{-last} \text{ (Cons - } xs) = pp\text{-last } xs$

The path fusion product (although we define it as a total function) should only be applied when the last element of the first argument is equal to the first element of the second argument.

primrec $pp\text{-fusion} :: 'a \text{ ppath} \Rightarrow 'a \text{ ppath} \Rightarrow 'a \text{ ppath}$ **where**

```

  pp-fusion (Node x) ys = ys
| pp-fusion (Cons x xs) ys = Cons x (pp-fusion xs ys)

```

We now go through the same steps as for traces and paths before, showing that the first and last element of a trace a left or right unit for that trace and that the fusion product on traces is associative.

lemma *pp-fusion-leftneutral* [simp]: $pp\text{-}fusion\ (Node\ (pp\text{-}first\ x))\ x = x$
by *simp*

lemma *pp-fusion-rightneutral* [simp]: $pp\text{-}fusion\ x\ (Node\ (pp\text{-}last\ x)) = x$
by (induct x) *simp-all*

lemma *pp-first-pp-fusion* [simp]:
 $pp\text{-}last\ x = pp\text{-}first\ y \implies pp\text{-}first\ (pp\text{-}fusion\ x\ y) = pp\text{-}first\ x$
by (induct x) *simp-all*

lemma *pp-last-pp-fusion* [simp]:
 $pp\text{-}last\ x = pp\text{-}first\ y \implies pp\text{-}last\ (pp\text{-}fusion\ x\ y) = pp\text{-}last\ y$
by (induct x) *simp-all*

lemma *pp-fusion-assoc* [simp]:
 $\llbracket pp\text{-}last\ x = pp\text{-}first\ y; pp\text{-}last\ y = pp\text{-}first\ z \rrbracket \implies pp\text{-}fusion\ x\ (pp\text{-}fusion\ y\ z)$
 $= pp\text{-}fusion\ (pp\text{-}fusion\ x\ y)\ z$
by (induct x) *simp-all*

We now lift the path fusion product to a complex product on sets of paths. This operation is total.

definition *pp-prod* :: 'a ppath set \Rightarrow 'a ppath set \Rightarrow 'a ppath set (**infixl** \cdot 70)
where $X \cdot Y = \{pp\text{-}fusion\ u\ v \mid u\ v.\ u \in X \wedge v \in Y \wedge pp\text{-}last\ u = pp\text{-}first\ v\}$

Next we define the set of paths of length one as the multiplicative unit of the path dioid.

definition *pp-one* :: 'a ppath set **where**
 $pp\text{-}one \equiv range\ Node$

We again provide an elimination rule.

lemma *pp-prod-iff*:
 $w \in X \cdot Y \iff (\exists u\ v.\ w = pp\text{-}fusion\ u\ v \wedge u \in X \wedge v \in Y \wedge pp\text{-}last\ u = pp\text{-}first\ v)$
by (unfold *pp-prod-def*) *auto*

interpretation *ppath-dioid*: *diod-one-zero* (\cup) (\cdot) *pp-one* $\{\}$ (\subseteq) (\subset)

proof

```

  fix x y z :: 'a ppath set
  show x  $\cup$  y  $\cup$  z = x  $\cup$  (y  $\cup$  z)
    by auto
  show x  $\cup$  y = y  $\cup$  x
    by auto

```

```

  show  $x \cdot y \cdot z = x \cdot (y \cdot z)$ 
    by (auto simp add: pp-prod-def, metis pp-first-pp-fusion pp-fusion-assoc, metis
pp-last-pp-fusion)
  show  $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$ 
    by (auto simp add: pp-prod-def)
  show  $pp-one \cdot x = x$ 
    by (auto simp add: pp-one-def pp-prod-def, metis pp-fusion.simps(1) pp-last.simps(1)
rangeI)
  show  $x \cdot pp-one = x$ 
    by (auto simp add: pp-one-def pp-prod-def, metis pp-first.simps(1) pp-fusion-rightneutral
rangeI)
  show  $\{\} \cup x = x$ 
    by auto
  show  $\{\} \cdot x = \{\}$ 
    by (simp add: pp-prod-def)
  show  $x \cdot \{\} = \{\}$ 
    by (simp add: pp-prod-def)
  show  $x \subseteq y \longleftrightarrow x \cup y = y$ 
    by auto
  show  $x \subset y \longleftrightarrow x \subseteq y \wedge x \neq y$ 
    by auto
  show  $x \cup x = x$ 
    by auto
  show  $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$ 
    by (auto simp add: pp-prod-def)
qed

no-notation
  pp-prod (infixl  $\cdot$  70)

```

0.12.9 The Distributive Lattice Dioid

A bounded distributive lattice is a distributive lattice with a least and a greatest element. Using Isabelle's lattice theory file we define a bounded distributive lattice as an axiomatic type class and show, using a sublocale statement, that every bounded distributive lattice is a dioid with one and zero.

```
class bounded-distributive-lattice = bounded-lattice + distrib-lattice
```

```
sublocale bounded-distributive-lattice  $\subseteq$  dioid-one-zero sup inf top bot less-eq
```

```
proof
```

```

  fix  $x \ y \ z$ 
  show  $\sup (\sup x \ y) \ z = \sup x (\sup y \ z)$ 
    by (fact sup-assoc)
  show  $\sup x \ y = \sup y \ x$ 
    by (fact sup-commute)
  show  $\inf (\inf x \ y) \ z = \inf x (\inf y \ z)$ 
    by (metis inf-commute inf.left-commute)

```



```

show  $\inf (\sup x y) z = \sup (\inf x z) (\inf y z)$ 
  by (fact inf-sup-distrib2)
show  $\inf \text{top } x = x$ 
  by simp
show  $\inf x \text{top} = x$ 
  by simp
show  $\sup \text{bot } x = x$ 
  by simp
show  $\inf \text{bot } x = \text{bot}$ 
  by simp
show  $\inf x \text{bot} = \text{bot}$ 
  by simp
show  $(x \leq y) = (\sup x y = y)$ 
  by (fact le-iff-sup)
show  $(x < y) = (x \leq y \wedge x \neq y)$ 
  by auto
show  $\sup x x = x$ 
  by simp
show  $\inf x (\sup y z) = \sup (\inf x y) (\inf x z)$ 
  by (fact inf-sup-distrib1)
qed

```

0.12.10 The Boolean Dioid

In this section we show that the booleans form a dioid, because the booleans form a bounded distributive lattice.

```

instantiation bool :: bounded-distributive-lattice
begin

```

```

  instance ..

```

```

end

```

```

interpretation boolean-dioid: dioid-one-zero sup inf True False less-eq less
  by (unfold-locales, simp-all add: inf-bool-def sup-bool-def)

```

0.12.11 The Max-Plus Dioid

The following dioids have important applications in combinatorial optimisations, control theory, algorithm design and computer networks.

A definition of reals extended with $+\infty$ and $-\infty$ may be found in *HOL/Library/Extended_Real.thy*. Alas, we require separate extensions with either $+\infty$ or $-\infty$.

The carrier set of the max-plus semiring is the set of real numbers extended by minus infinity. The operation of addition is maximum, the operation of multiplication is addition, the additive unit is minus infinity and the multiplicative unit is zero.

datatype *mreal* = *mreal* *real* | *MInfty* — minus infinity

fun *mreal-max* **where**
 mreal-max (*mreal* *x*) (*mreal* *y*) = *mreal* (*max* *x* *y*)
 | *mreal-max* *x* *MInfty* = *x*
 | *mreal-max* *MInfty* *y* = *y*

lemma *mreal-max-simp-3* [*simp*]: *mreal-max* *MInfty* *y* = *y*
by (*cases* *y*, *simp-all*)

fun *mreal-plus* **where**
 mreal-plus (*mreal* *x*) (*mreal* *y*) = *mreal* (*x* + *y*)
 | *mreal-plus* - - = *MInfty*

We now show that the max plus-semiring satisfies the axioms of selective semirings, from which it follows that it satisfies the dioid axioms.

instantiation *mreal* :: *selective-semiring*
begin

definition *zero-mreal-def*:
 $0 \equiv \text{MInfty}$

definition *one-mreal-def*:
 $1 \equiv \text{mreal } 0$

definition *plus-mreal-def*:
 $x + y \equiv \text{mreal-max } x \ y$

definition *times-mreal-def*:
 $x * y \equiv \text{mreal-plus } x \ y$

definition *less-eq-mreal-def*:
 $(x::\text{mreal}) \leq y \equiv x + y = y$

definition *less-mreal-def*:
 $(x::\text{mreal}) < y \equiv x \leq y \wedge x \neq y$

instance

proof

fix *x y z* :: *mreal*
show $x + y + z = x + (y + z)$
 by (*cases* *x*, *cases* *y*, *cases* *z*, *simp-all* *add*: *plus-mreal-def*)
show $x + y = y + x$
 by (*cases* *x*, *cases* *y*, *simp-all* *add*: *plus-mreal-def*)
show $x * y * z = x * (y * z)$
 by (*cases* *x*, *cases* *y*, *cases* *z*, *simp-all* *add*: *times-mreal-def*)
show $(x + y) * z = x * z + y * z$
 by (*cases* *x*, *cases* *y*, *cases* *z*, *simp-all* *add*: *plus-mreal-def* *times-mreal-def*)
show $1 * x = x$

```

    by (cases x, simp-all add: one-mreal-def times-mreal-def)
  show  $x * 1 = x$ 
    by (cases x, simp-all add: one-mreal-def times-mreal-def)
  show  $0 + x = x$ 
    by (cases x, simp-all add: plus-mreal-def zero-mreal-def)
  show  $0 * x = 0$ 
    by (cases x, simp-all add: times-mreal-def zero-mreal-def)
  show  $x * 0 = 0$ 
    by (cases x, simp-all add: times-mreal-def zero-mreal-def)
  show  $x \leq y \iff x + y = y$ 
    by (metis less-eq-mreal-def)
  show  $x < y \iff x \leq y \wedge x \neq y$ 
    by (metis less-mreal-def)
  show  $x + y = x \vee x + y = y$ 
    by (cases x, cases y, simp-all add: plus-mreal-def, metis linorder-le-cases
max.absorb-iff2 max.absorb1)
  show  $x * (y + z) = x * y + x * z$ 
    by (cases x, cases y, cases z, simp-all add: plus-mreal-def times-mreal-def)
qed
end

```

0.12.12 The Min-Plus Dioid

The min-plus dioid is also known as *tropical semiring*. Here we need to add a positive infinity to the real numbers. The procedure follows that of max-plus semirings.

datatype *preal* = *preal real* | *PInf* — plus infinity

fun *preal-min* **where**

```

  preal-min (preal x) (preal y) = preal (min x y)
| preal-min x PInf = x
| preal-min PInf y = y

```

lemma *preal-min-simp-3* [simp]: *preal-min PInf y = y*

by (cases y, simp-all)

fun *preal-plus* **where**

```

  preal-plus (preal x) (preal y) = preal (x + y)
| preal-plus - - = PInf

```

instantiation *preal* :: *selective-semiring*

begin

definition *zero-preal-def*:

$0 \equiv PInf$

definition *one-preal-def*:

$1 \equiv preal\ 0$

definition *plus-preal-def*:

$$x + y \equiv \text{preal-min } x \ y$$

definition *times-preal-def*:

$$x * y \equiv \text{preal-plus } x \ y$$

definition *less-eq-preal-def*:

$$(x::\text{preal}) \leq y \equiv x + y = y$$

definition *less-preal-def*:

$$(x::\text{preal}) < y \equiv x \leq y \wedge x \neq y$$

instance

proof

fix $x \ y \ z :: \text{preal}$

show $x + y + z = x + (y + z)$

by (*cases x, cases y, cases z, simp-all add: plus-preal-def*)

show $x + y = y + x$

by (*cases x, cases y, simp-all add: plus-preal-def*)

show $x * y * z = x * (y * z)$

by (*cases x, cases y, cases z, simp-all add: times-preal-def*)

show $(x + y) * z = x * z + y * z$

by (*cases x, cases y, cases z, simp-all add: plus-preal-def times-preal-def*)

show $1 * x = x$

by (*cases x, simp-all add: one-preal-def times-preal-def*)

show $x * 1 = x$

by (*cases x, simp-all add: one-preal-def times-preal-def*)

show $0 + x = x$

by (*cases x, simp-all add: plus-preal-def zero-preal-def*)

show $0 * x = 0$

by (*cases x, simp-all add: times-preal-def zero-preal-def*)

show $x * 0 = 0$

by (*cases x, simp-all add: times-preal-def zero-preal-def*)

show $x \leq y \longleftrightarrow x + y = y$

by (*metis less-eq-preal-def*)

show $x < y \longleftrightarrow x \leq y \wedge x \neq y$

by (*metis less-preal-def*)

show $x + y = x \vee x + y = y$

by (*cases x, cases y, simp-all add: plus-preal-def, metis linorder-le-cases min.absorb2 min.absorb-iff1*)

show $x * (y + z) = x * y + x * z$

by (*cases x, cases y, cases z, simp-all add: plus-preal-def times-preal-def*)

qed

end

Variants of min-plus and max-plus semirings can easily be obtained. Here we formalise the min-plus semiring over the natural numbers as an example.

datatype *pnat* = *pnat nat* | *PInfty* — plus infinity

fun *pnat-min* **where**
 pnat-min (*pnat x*) (*pnat y*) = *pnat (min x y)*
 | *pnat-min x PInfty* = *x*
 | *pnat-min PInfty x* = *x*

lemma *pnat-min-simp-3* [*simp*]: *pnat-min PInfty y* = *y*
by (*cases y, simp-all*)

fun *pnat-plus* **where**
 pnat-plus (*pnat x*) (*pnat y*) = *pnat (x + y)*
 | *pnat-plus - -* = *PInfty*

instantiation *pnat* :: *selective-semiring*
begin

definition *zero-pnat-def*:
 $0 \equiv PInfty$

definition *one-pnat-def*:
 $1 \equiv pnat\ 0$

definition *plus-pnat-def*:
 $x + y \equiv pnat-min\ x\ y$

definition *times-pnat-def*:
 $x * y \equiv pnat-plus\ x\ y$

definition *less-eq-pnat-def*:
 $(x::pnat) \leq y \equiv x + y = y$

definition *less-pnat-def*:
 $(x::pnat) < y \equiv x \leq y \wedge x \neq y$

lemma *zero-pnat-top*: $(x::pnat) \leq 1$
by (*cases x, simp-all add: less-eq-pnat-def plus-pnat-def one-pnat-def*)

instance

proof

fix *x y z* :: *pnat*
 show $x + y + z = x + (y + z)$
 by (*cases x, cases y, cases z, simp-all add: plus-pnat-def*)
 show $x + y = y + x$
 by (*cases x, cases y, simp-all add: plus-pnat-def*)
 show $x * y * z = x * (y * z)$
 by (*cases x, cases y, cases z, simp-all add: times-pnat-def*)
 show $(x + y) * z = x * z + y * z$
 by (*cases x, cases y, cases z, simp-all add: plus-pnat-def times-pnat-def*)

```

show  $1 * x = x$ 
  by (cases x, simp-all add: one-pnat-def times-pnat-def)
show  $x * 1 = x$ 
  by (cases x, simp-all add: one-pnat-def times-pnat-def)
show  $0 + x = x$ 
  by (cases x, simp-all add: plus-pnat-def zero-pnat-def)
show  $0 * x = 0$ 
  by (cases x, simp-all add: times-pnat-def zero-pnat-def)
show  $x * 0 = 0$ 
  by (cases x, simp-all add: times-pnat-def zero-pnat-def)
show  $x \leq y \longleftrightarrow x + y = y$ 
  by (metis less-eq-pnat-def)
show  $x < y \longleftrightarrow x \leq y \wedge x \neq y$ 
  by (metis less-pnat-def)
show  $x + y = x \vee x + y = y$ 
  by (cases x, cases y, simp-all add: plus-pnat-def, metis linorder-le-cases
min.absorb2 min.absorb-iff1)
show  $x * (y + z) = x * y + x * z$ 
  by (cases x, cases y, cases z, simp-all add: plus-pnat-def times-pnat-def)
qed

end

end

```

0.13 Models of Kleene Algebras

```

theory Kleene-Algebra-Models
imports Kleene-Algebra Dioid-Models
begin

```

We now show that most of the models considered for dioids are also Kleene algebras. Some of the dioid models cannot be expanded, for instance max-plus and min-plus semirings, but we do not formalise this fact. We also currently do not show that formal powerseries and matrices form Kleene algebras.

The interpretation proofs for some of the following models are quite similar. One could, perhaps, abstract out common reasoning in the future.

0.13.1 Preliminary Lemmas

We first prove two induction-style statements for dioids that are useful for establishing the full induction laws. In the future these will live in a theory file on finite sums for Kleene algebras.

```

context dioid-one-zero
begin

```

```

lemma power-inductl:  $z + x \cdot y \leq y \implies (x \wedge n) \cdot z \leq y$ 
proof (induct n)
  case 0 show ?case
    using 0.prem1 by auto
  case Suc thus ?case
    by (auto, metis mult.assoc mult-isol order-trans)
qed

```

```

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.prem1 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

```

0.13.2 The Powerset Kleene Algebra over a Monoid

We now show that the powerset dioid forms a Kleene algebra. The Kleene star is defined as in language theory.

```

lemma Un-0-Suc:  $(\bigcup n. f \cdot n) = f \cdot 0 \cup (\bigcup n. f \cdot (\text{Suc } n))$ 
by auto (metis not0-implies-Suc)

```

0.13.3 Relation Kleene Algebras

We now show that binary relations form Kleene algebras. While we could have used the reflexive transitive closure operation as the Kleene star, we prefer the equivalent definition of the star as the sum of powers. This essentially allows us to copy previous proofs.

```

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

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

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

```

0.13.4 Trace Kleene Algebras

Again, the proof that sets of traces form Kleene algebras follows the same schema.

definition *t-star* :: $('p, 'a) \text{ trace set} \Rightarrow ('p, 'a) \text{ trace set}$ **where**
 $t\text{-star } X \equiv \bigcup n. \text{trace-diod.power } X \ n$

lemma *t-star-elim*: $x \in t\text{-star } X \longleftrightarrow (\exists n. x \in \text{trace-diod.power } X \ n)$
by (*simp* *add*: *t-star-def*)

lemma *t-star-contl*: $t\text{-prod } X \ (t\text{-star } Y) = (\bigcup n. t\text{-prod } X \ (\text{trace-diod.power } Y \ n))$
by (*auto* *simp* *add*: *t-star-elim* *t-prod-def*)

lemma *t-star-contr*: $t\text{-prod } (t\text{-star } X) \ Y = (\bigcup n. t\text{-prod } (\text{trace-diod.power } X \ n) \ Y)$


```

Y)
  by (auto simp add: t-star-elim t-prod-def)

interpretation trace-kleene-algebra: kleene-algebra ( $\cup$ ) t-prod t-one t-zero ( $\subseteq$ ) ( $\subset$ )
t-star
proof
  fix X Y Z :: ('a, 'b) trace set
  show t-one  $\cup$  t-prod X (t-star X)  $\subseteq$  t-star X
  proof –
    have t-one  $\cup$  t-prod X (t-star X) = (trace-dioid.power X 0)  $\cup$  ( $\bigcup n.$ 
trace-dioid.power X (Suc n))
    by (auto simp add: t-star-def t-prod-def)
    also have ... = ( $\bigcup n.$  trace-dioid.power X n)
    by (metis Un-0-Suc)
    also have ... = t-star X
    by (metis t-star-def)
    finally show ?thesis
    by (metis subset-refl)
  qed
  show Z  $\cup$  t-prod X Y  $\subseteq$  Y  $\implies$  t-prod (t-star X) Z  $\subseteq$  Y
  by (simp only: ball-UNIV t-star-contr SUP-le-iff) (metis trace-dioid.power-inductl)
  show Z  $\cup$  t-prod Y X  $\subseteq$  Y  $\implies$  t-prod Z (t-star X)  $\subseteq$  Y
  by (simp only: ball-UNIV t-star-contrl SUP-le-iff) (metis trace-dioid.power-inductr)
qed

```

0.13.5 Path Kleene Algebras

We start with paths that include the empty path.

definition $p\text{-star} :: 'a \text{ path set} \Rightarrow 'a \text{ path set}$ **where**

$$p\text{-star } X \equiv \bigcup n. \text{path-dioid.power } X \ n$$

lemma $p\text{-star-elim}: x \in p\text{-star } X \longleftrightarrow (\exists n. x \in \text{path-dioid.power } X \ n)$

by (simp add: p-star-def)

lemma $p\text{-star-contrl}: p\text{-prod } X \ (p\text{-star } Y) = (\bigcup n. p\text{-prod } X \ (\text{path-dioid.power } Y \ n))$

apply (auto simp add: p-prod-def p-star-elim)

apply (metis p-fusion.simps(1))

apply metis

apply (metis p-fusion.simps(1) p-star-elim)

apply (metis p-star-elim)

done

lemma $p\text{-star-contr}: p\text{-prod } (p\text{-star } X) \ Y = (\bigcup n. p\text{-prod } (\text{path-dioid.power } X \ n) \ Y)$

apply (auto simp add: p-prod-def p-star-elim)

apply (metis p-fusion.simps(1))

apply metis

apply (metis p-fusion.simps(1) p-star-elim)

apply (*metis p-star-elim*)
done

interpretation *path-kleene-algebra*: *kleene-algebra* (\cup) *p-prod* *p-one* $\{\}$ (\subseteq) (\subset)
p-star

proof

fix $X Y Z :: 'a \text{ path set}$
show $p\text{-one} \cup p\text{-prod } X \ (p\text{-star } X) \subseteq p\text{-star } X$
proof –
have $p\text{-one} \cup p\text{-prod } X \ (p\text{-star } X) = (\text{path-diod.power } X \ 0) \cup (\bigcup n. \text{path-diod.power } X \ (\text{Suc } n))$
by (*auto simp add: p-star-def p-prod-def*)
also have $\dots = (\bigcup n. \text{path-diod.power } X \ n)$
by (*metis Un-0-Suc*)
also have $\dots = p\text{-star } X$
by (*metis p-star-def*)
finally show *?thesis*
by (*metis subset-refl*)
qed
show $Z \cup p\text{-prod } X \ Y \subseteq Y \implies p\text{-prod } (p\text{-star } X) \ Z \subseteq Y$
by (*simp only: ball-UNIV p-star-contr SUP-le-iff*) (*metis path-diod.power-inductl*)
show $Z \cup p\text{-prod } Y \ X \subseteq Y \implies p\text{-prod } Z \ (p\text{-star } X) \subseteq Y$
by (*simp only: ball-UNIV p-star-contrl SUP-le-iff*) (*metis path-diod.power-inductr*)
qed

We now consider a notion of paths that does not include the empty path.

definition *pp-star* :: $'a \text{ ppath set} \Rightarrow 'a \text{ ppath set}$ **where**
 $pp\text{-star } X \equiv \bigcup n. \text{ppath-diod.power } X \ n$

lemma *pp-star-elim*: $x \in pp\text{-star } X \longleftrightarrow (\exists n. x \in \text{ppath-diod.power } X \ n)$
by (*simp add: pp-star-def*)

lemma *pp-star-contrl*: $pp\text{-prod } X \ (pp\text{-star } Y) = (\bigcup n. pp\text{-prod } X \ (\text{ppath-diod.power } Y \ n))$
by (*auto simp add: pp-prod-def pp-star-elim*)

lemma *pp-star-contr*: $pp\text{-prod } (pp\text{-star } X) \ Y = (\bigcup n. pp\text{-prod } (\text{ppath-diod.power } X \ n) \ Y)$
by (*auto simp add: pp-prod-def pp-star-elim*)

interpretation *ppath-kleene-algebra*: *kleene-algebra* (\cup) *pp-prod* *pp-one* $\{\}$ (\subseteq) (\subset)
pp-star

proof

fix $X Y Z :: 'a \text{ ppath set}$
show $pp\text{-one} \cup pp\text{-prod } X \ (pp\text{-star } X) \subseteq pp\text{-star } X$
proof –
have $pp\text{-one} \cup pp\text{-prod } X \ (pp\text{-star } X) = (\text{ppath-diod.power } X \ 0) \cup (\bigcup n. \text{ppath-diod.power } X \ (\text{Suc } n))$
by (*auto simp add: pp-star-def pp-prod-def*)

```

    also have ... = ( $\bigcup n. \text{ppath-diod.power } X \ n$ )
      by (metis Un-0-Suc)
    also have ... =  $\text{pp-star } X$ 
      by (metis pp-star-def)
    finally show ?thesis
      by (metis subset-refl)
  qed
show  $Z \cup \text{pp-prod } X \ Y \subseteq Y \implies \text{pp-prod } (\text{pp-star } X) \ Z \subseteq Y$ 
  by (simp only: ball-UNIV pp-star-contr SUP-le-iff) (metis ppath-diod.power-inductl)
show  $Z \cup \text{pp-prod } Y \ X \subseteq Y \implies \text{pp-prod } Z \ (\text{pp-star } X) \subseteq Y$ 
  by (simp only: ball-UNIV pp-star-contr SUP-le-iff) (metis ppath-diod.power-inductr)
qed

```

0.13.6 The Distributive Lattice Kleene Algebra

In the case of bounded distributive lattices, the star maps all elements to to the maximal element.

definition (in *bounded-distributive-lattice*) $\text{bdl-star} :: 'a \Rightarrow 'a$ **where**
 $\text{bdl-star } x = \text{top}$

sublocale *bounded-distributive-lattice* \subseteq *kleene-algebra* *sup inf top bot less-eq less*
 bdl-star

proof

```

  fix x y z :: 'a
  show  $\text{sup top } (\text{inf } x \ (\text{bdl-star } x)) \leq \text{bdl-star } x$ 
    by (simp add: bdl-star-def)
  show  $\text{sup } z \ (\text{inf } x \ y) \leq y \implies \text{inf } (\text{bdl-star } x) \ z \leq y$ 
    by (simp add: bdl-star-def)
  show  $\text{sup } z \ (\text{inf } y \ x) \leq y \implies \text{inf } z \ (\text{bdl-star } x) \leq y$ 
    by (simp add: bdl-star-def)
  qed

```

0.13.7 The Min-Plus Kleene Algebra

One cannot define a Kleene star for max-plus and min-plus algebras that range over the real numbers. Here we define the star for a min-plus algebra restricted to natural numbers and $+\infty$. The resulting Kleene algebra is commutative. Similar variants can be obtained for max-plus algebras and other algebras ranging over the positive or negative integers.

instantiation $\text{pnat} :: \text{commutative-kleene-algebra}$
begin

definition star-pnat **where**
 $x^* \equiv (1::\text{pnat})$

instance
proof

```

    fix x y z :: pnat
    show 1 + x · x* ≤ x*
      by (metis star-pnat-def zero-pnat-top)
    show z + x · y ≤ y ⇒ x* · z ≤ y
      by (simp add: star-pnat-def)
    show z + y · x ≤ y ⇒ z · x* ≤ y
      by (simp add: star-pnat-def)
    show x · y = y · x
      unfolding times-pnat-def by (cases x, cases y, simp-all)
  qed

end

end

```

0.14 Domain Semirings

```

theory Domain-Semiring
imports Kleene-Algebra

```

```

begin

```

0.14.1 Domain Semigroups and Domain Monoids

```

class domain-op =
  fixes domain-op :: 'a ⇒ 'a (d)

```

First we define the class of domain semigroups. Axioms are taken from [?].

```

class domain-semigroup = semigroup-mult + domain-op +
  assumes dsg1 [simp]: d x · x = x
  and dsg2 [simp]: d (x · d y) = d (x · y)
  and dsg3 [simp]: d (d x · y) = d x · d y
  and dsg4: d x · d y = d y · d x

```

```

begin

```

```

lemma domain-invol [simp]: d (d x) = d x
proof -
  have d (d x) = d (d (d x · x))
    by simp
  also have ... = d (d x · d x)
    using dsg3 by presburger
  also have ... = d (d x · x)
    by simp
  finally show ?thesis
    by simp
qed

```

The next lemmas show that domain elements form semilattices.

lemma *dom-el-idem* [*simp*]: $d\ x \cdot d\ x = d\ x$

proof –

have $d\ x \cdot d\ x = d\ (d\ x \cdot x)$

using *dsg3* **by** *presburger*

thus *?thesis*

by *simp*

qed

lemma *dom-mult-closed* [*simp*]: $d\ (d\ x \cdot d\ y) = d\ x \cdot d\ y$

by *simp*

lemma *dom-lc3* [*simp*]: $d\ x \cdot d\ (x \cdot y) = d\ (x \cdot y)$

proof –

have $d\ x \cdot d\ (x \cdot y) = d\ (d\ x \cdot x \cdot y)$

using *dsg3 mult-assoc* **by** *presburger*

thus *?thesis*

by *simp*

qed

lemma *d-fixpoint*: $(\exists y. x = d\ y) \longleftrightarrow x = d\ x$

by *auto*

lemma *d-type*: $\forall P. (\forall x. x = d\ x \longrightarrow P\ x) \longleftrightarrow (\forall x. P\ (d\ x))$

by (*metis domain-invol*)

We define the semilattice ordering on domain semigroups and explore the semilattice of domain elements from the order point of view.

definition *ds-ord* :: $'a \Rightarrow 'a \Rightarrow \text{bool}$ (**infix** \sqsubseteq 50) **where**

$x \sqsubseteq y \longleftrightarrow x = d\ x \cdot y$

lemma *ds-ord-reft*: $x \sqsubseteq x$

by (*simp add: ds-ord-def*)

lemma *ds-ord-trans*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$

proof –

assume $x \sqsubseteq y$ **and** $a: y \sqsubseteq z$

hence $b: x = d\ x \cdot y$

using *ds-ord-def* **by** *blast*

hence $x = d\ x \cdot d\ y \cdot z$

using *a ds-ord-def mult-assoc* **by** *force*

also have $\dots = d\ (d\ x \cdot y) \cdot z$

by *simp*

also have $\dots = d\ x \cdot z$

using *b* **by** *auto*

finally show *?thesis*

using *ds-ord-def* **by** *blast*

qed

lemma *ds-ord-antisym*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$

proof –
assume $a: x \sqsubseteq y$ **and** $y \sqsubseteq x$
hence $b: y = d\ y \cdot x$
using *ds-ord-def* **by** *auto*
have $x = d\ x \cdot d\ y \cdot x$
using $a\ b\ ds\text{-ord-def}\ mult\text{-assoc}$ **by** *force*
also have $\dots = d\ y \cdot x$
by (*metis* (*full-types*) $b\ ds\ g3\ ds\ g4$)
thus *?thesis*
using b *calculation* **by** *presburger*
qed

This relation is indeed an order.

sublocale $ds: order\ (\sqsubseteq)\ \lambda x\ y. (x \sqsubseteq y \wedge x \neq y)$
proof
show $\bigwedge x\ y. (x \sqsubseteq y \wedge x \neq y) = (x \sqsubseteq y \wedge \neg y \sqsubseteq x)$
using *ds-ord-antisym* **by** *blast*
show $\bigwedge x. x \sqsubseteq x$
by (*rule ds-ord-refl*)
show $\bigwedge x\ y\ z. x \sqsubseteq y \implies y \sqsubseteq z \implies x \sqsubseteq z$
by (*rule ds-ord-trans*)
show $\bigwedge x\ y. x \sqsubseteq y \implies y \sqsubseteq x \implies x = y$
by (*rule ds-ord-antisym*)
qed

lemma *ds-ord-eq*: $x \sqsubseteq d\ x \longleftrightarrow x = d\ x$
by (*simp add: ds-ord-def*)

lemma $x \sqsubseteq y \implies z \cdot x \sqsubseteq z \cdot y$

oops

lemma *ds-ord-iso-right*: $x \sqsubseteq y \implies x \cdot z \sqsubseteq y \cdot z$
proof –
assume $x \sqsubseteq y$
hence $a: x = d\ x \cdot y$
by (*simp add: ds-ord-def*)
hence $x \cdot z = d\ x \cdot y \cdot z$
by *auto*
also have $\dots = d\ (d\ x \cdot y \cdot z) \cdot d\ x \cdot y \cdot z$
using *dsg1 mult-assoc* **by** *presburger*
also have $\dots = d\ (x \cdot z) \cdot d\ x \cdot y \cdot z$
using a **by** *presburger*
finally show *?thesis*
using *ds-ord-def dsg4 mult-assoc* **by** *auto*
qed

The order on domain elements could as well be defined based on multiplication/meet.

lemma *ds-ord-sl-ord*: $d\ x \sqsubseteq d\ y \longleftrightarrow d\ x \cdot d\ y = d\ x$
using *ds-ord-def* **by** *auto*

lemma *ds-ord-1*: $d\ (x \cdot y) \sqsubseteq d\ x$
by (*simp add: ds-ord-sl-ord dsq4*)

lemma *ds-subid-aux*: $d\ x \cdot y \sqsubseteq y$
by (*simp add: ds-ord-def mult-assoc*)

lemma $y \cdot d\ x \sqsubseteq y$

oops

lemma *ds-dom-iso*: $x \sqsubseteq y \implies d\ x \sqsubseteq d\ y$

proof –

assume $x \sqsubseteq y$
hence $x = d\ x \cdot y$
by (*simp add: ds-ord-def*)
hence $d\ x = d\ (d\ x \cdot y)$
by *presburger*
also have $\dots = d\ x \cdot d\ y$
by *simp*
finally show *?thesis*
using *ds-ord-sl-ord* **by** *auto*

qed

lemma *ds-dom-llp*: $x \sqsubseteq d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$

proof

assume $x \sqsubseteq d\ y \cdot x$
hence $x = d\ y \cdot x$
by (*simp add: ds-subid-aux ds.order.antisym*)
hence $d\ x = d\ (d\ y \cdot x)$
by *presburger*
thus $d\ x \sqsubseteq d\ y$
using *ds-ord-sl-ord dsq4* **by** *force*

next

assume $d\ x \sqsubseteq d\ y$
thus $x \sqsubseteq d\ y \cdot x$
by (*metis (no-types) ds-ord-iso-right dsq1*)

qed

lemma *ds-dom-llp-strong*: $x = d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$
by (*simp add: ds-dom-llp ds.eq-iff ds-subid-aux*)

definition *refines* :: $'a \Rightarrow 'a \Rightarrow \text{bool}$

where *refines* $x\ y \equiv d\ y \sqsubseteq d\ x \wedge (d\ y) \cdot x \sqsubseteq y$

lemma *refines-refl*: *refines* $x\ x$
using *refines-def* **by** *simp*

lemma *refines-trans*: $\text{refines } x \ y \implies \text{refines } y \ z \implies \text{refines } x \ z$
unfolding *refines-def*
by (*metis domain-invol ds.dual-order.trans dsg1 dsg3 ds-ord-def*)

lemma *refines-antisym*: $\text{refines } x \ y \implies \text{refines } y \ x \implies x = y$
unfolding *refines-def*
using *ds-dom-llp ds-ord-antisym* **by** *fastforce*

sublocale *ref*: *order* *refines* $\lambda x \ y. (\text{refines } x \ y \wedge x \neq y)$
proof
show $\bigwedge x \ y. (\text{refines } x \ y \wedge x \neq y) = (\text{refines } x \ y \wedge \neg \text{refines } y \ x)$
using *refines-antisym* **by** *blast*
show $\bigwedge x. \text{refines } x \ x$
by (*rule refines-refl*)
show $\bigwedge x \ y \ z. \text{refines } x \ y \implies \text{refines } y \ z \implies \text{refines } x \ z$
by (*rule refines-trans*)
show $\bigwedge x \ y. \text{refines } x \ y \implies \text{refines } y \ x \implies x = y$
by (*rule refines-antisym*)
qed
end

We expand domain semigroups to domain monoids.

class *domain-monoid* = *monoid-mult* + *domain-semigroup*
begin

lemma *dom-one* [*simp*]: $d \ 1 = 1$
proof –
have $1 = d \ 1 \cdot 1$
using *dsg1* **by** *presburger*
thus *?thesis*
by *simp*
qed

lemma *ds-subid-eq*: $x \sqsubseteq 1 \longleftrightarrow x = d \ x$
by (*simp add: ds-ord-def*)

end

0.14.2 Domain Near-Semirings

The axioms for domain near-semirings are taken from [?].

class *domain-near-semiring* = *ab-near-semiring* + *plus-ord* + *domain-op* +
assumes *dns1* [*simp*]: $d \ x \cdot x = x$
and *dns2* [*simp*]: $d \ (x \cdot d \ y) = d(x \cdot y)$
and *dns3* [*simp*]: $d \ (x + y) = d \ x + d \ y$
and *dns4*: $d \ x \cdot d \ y = d \ y \cdot d \ x$
and *dns5* [*simp*]: $d \ x \cdot (d \ x + d \ y) = d \ x$

begin

Domain near-semirings are automatically dioids; addition is idempotent.

```

subclass near-dioid
proof
  show  $\bigwedge x. x + x = x$ 
  proof –
    fix  $x$ 
    have  $a: d\ x = d\ x \cdot d\ (x + x)$ 
      using dns3 dns5 by presburger
    have  $d\ (x + x) = d\ (x + x + (x + x)) \cdot d\ (x + x)$ 
      by (metis (no-types) dns3 dns4 dns5)
    hence  $d\ (x + x) = d\ (x + x) + d\ (x + x)$ 
      by simp
    thus  $x + x = x$ 
      by (metis a dns1 dns4 distrib-right')
  qed
qed

```

Next we prepare to show that domain near-semirings are domain semigroups.

```

lemma dom-iso:  $x \leq y \implies d\ x \leq d\ y$ 
  using order-prop by auto

```

```

lemma dom-add-closed [simp]:  $d\ (d\ x + d\ y) = d\ x + d\ y$ 
proof –
  have  $d\ (d\ x + d\ y) = d\ (d\ x) + d\ (d\ y)$ 
    by simp
  thus ?thesis
    by (metis dns1 dns2 dns3 dns4)
qed

```

```

lemma dom-absorp-2 [simp]:  $d\ x + d\ x \cdot d\ y = d\ x$ 
proof –
  have  $d\ x + d\ x \cdot d\ y = d\ x \cdot d\ x + d\ x \cdot d\ y$ 
    by (metis add-idem' dns5)
  also have  $\dots = (d\ x + d\ y) \cdot d\ x$ 
    by (simp add: dns4)
  also have  $\dots = d\ x \cdot (d\ x + d\ y)$ 
    by (metis dom-add-closed dns4)
  finally show ?thesis
    by simp
qed

```

```

lemma dom-1:  $d\ (x \cdot y) \leq d\ x$ 
proof –
  have  $d\ (x \cdot y) = d\ (d\ x \cdot d\ (x \cdot y))$ 
    by (metis dns1 dns2 mult-assoc)
  also have  $\dots \leq d\ (d\ x) + d\ (d\ x \cdot d\ (x \cdot y))$ 

```

```

    by simp
  also have ... = d (d x + d x · d (x · y))
    using dns3 by presburger
  also have ... = d (d x)
    by simp
  finally show ?thesis
    by (metis dom-add-closed add-idem')
qed

```

```

lemma dom-subid-aux2: d x · y ≤ y
proof -
  have d x · y ≤ d (x + d y) · y
    by (simp add: mult-isor)
  also have ... = (d x + d (d y)) · d y · y
    using dns1 dns3 mult-assoc by presburger
  also have ... = (d y + d y · d x) · y
    by (simp add: dns4 add-commute)
  finally show ?thesis
    by simp
qed

```

```

lemma dom-glb: d x ≤ d y ⇒ d x ≤ d z ⇒ d x ≤ d y · d z
  by (metis dns5 less-eq-def mult-isor)

```

```

lemma dom-glb-eq: d x ≤ d y · d z ⇔ d x ≤ d y ∧ d x ≤ d z
proof -
  have d x ≤ d z → d x ≤ d z
    by meson
  then show ?thesis
    by (metis (no-types) dom-absorp-2 dom-glb dom-subid-aux2 local.dual-order.trans
    local.join.sup.coboundedI2)
qed

```

```

lemma dom-ord: d x ≤ d y ⇔ d x · d y = d x
proof
  assume d x ≤ d y
  hence d x + d y = d y
    by (simp add: less-eq-def)
  thus d x · d y = d x
    by (metis dns5)
next
  assume d x · d y = d x
  thus d x ≤ d y
    by (metis dom-subid-aux2)
qed

```

```

lemma dom-export [simp]: d (d x · y) = d x · d y
proof (rule antisym)
  have d (d x · y) = d (d (d x · y)) · d (d x · y)

```

```

    using dns1 by presburger
  also have ... = d (d x · d y) · d (d x · y)
    by (metis dns1 dns2 mult-assoc)
  finally show a: d (d x · y) ≤ d x · d y
    by (metis (no-types) dom-add-closed dom-glb dom-1 add-idem' dns2 dns4)
  have d (d x · y) = d (d x · y) · d x
    using a dom-glb-eq dom-ord by force
  hence d x · d y = d (d x · y) · d y
    by (metis dns1 dns2 mult-assoc)
  thus d x · d y ≤ d (d x · y)
    using a dom-glb-eq dom-ord by auto
qed

```

```

subclass domain-semigroup
by (unfold-locales, auto simp: dns4)

```

We compare the domain semigroup ordering with that of the dioid.

```

lemma d-two-orders: d x ⊆ d y ⟷ d x ≤ d y
  by (simp add: dom-ord ds-ord-sl-ord)

```

```

lemma two-orders: x ⊆ y ⟹ x ≤ y
  by (metis dom-subid-aux2 ds-ord-def)

```

```

lemma x ≤ y ⟹ x ⊆ y

```

oops

Next we prove additional properties.

```

lemma dom-subdist: d x ≤ d (x + y)
  by simp

```

```

lemma dom-distrib: d x + d y · d z = (d x + d y) · (d x + d z)

```

proof –

```

  have (d x + d y) · (d x + d z) = d x · (d x + d z) + d y · (d x + d z)
    using distrib-right' by blast
  also have ... = d x + (d x + d z) · d y
    by (metis (no-types) dns3 dns5 dsg4)
  also have ... = d x + d x · d y + d z · d y
    using add-assoc' distrib-right' by presburger
  finally show ?thesis
    by (simp add: dsg4)

```

qed

```

lemma dom-llp1: x ≤ d y · x ⟹ d x ≤ d y

```

proof –

```

  assume x ≤ d y · x
  hence d x ≤ d (d y · x)
    using dom-iso by blast
  also have ... = d y · d x

```

```

    by simp
  finally show  $d\ x \leq d\ y$ 
    by (simp add: dom-glb-eq)
qed

```

```

lemma dom-llp2:  $d\ x \leq d\ y \implies x \leq d\ y \cdot x$ 
  using d-two-orders local.ds-dom-llp two-orders by blast

```

```

lemma dom-llp:  $x \leq d\ y \cdot x \longleftrightarrow d\ x \leq d\ y$ 
  using dom-llp1 dom-llp2 by blast

```

```

end

```

We expand domain near-semirings by an additive unit, using slightly different axioms.

```

class domain-near-semiring-one = ab-near-semiring-one + plus-ord + domain-op
+
  assumes dns01 [simp]:  $x + d\ x \cdot x = d\ x \cdot x$ 
  and dns02 [simp]:  $d\ (x \cdot d\ y) = d\ (x \cdot y)$ 
  and dns03 [simp]:  $d\ x + 1 = 1$ 
  and dns04 [simp]:  $d\ (x + y) = d\ x + d\ y$ 
  and dns05:  $d\ x \cdot d\ y = d\ y \cdot d\ x$ 

```

```

begin

```

The previous axioms are derivable.

```

subclass domain-near-semiring
proof
  show  $a: \bigwedge x. d\ x \cdot x = x$ 
    by (metis add-commute local.dns03 local.distrib-right' local.dns01 local.mult-onel)
  show  $\bigwedge x\ y. d\ (x \cdot d\ y) = d\ (x \cdot y)$ 
    by simp
  show  $\bigwedge x\ y. d\ (x + y) = d\ x + d\ y$ 
    by simp
  show  $\bigwedge x\ y. d\ x \cdot d\ y = d\ y \cdot d\ x$ 
    by (simp add: dns05)
  show  $\bigwedge x\ y. d\ x \cdot (d\ x + d\ y) = d\ x$ 
  proof -
    fix  $x\ y$ 
    have  $\bigwedge x. 1 + d\ x = 1$ 
      using add-commute dns03 by presburger
    thus  $d\ x \cdot (d\ x + d\ y) = d\ x$ 
      by (metis (no-types) a dns02 dns04 dns05 distrib-right' mult-onel)
  qed
qed

```

```

subclass domain-monoid ..

```

```

lemma dom-subid:  $d\ x \leq 1$ 

```

by (*simp add: less-eq-def*)

end

We add a left unit of multiplication.

class *domain-near-semiring-one-zero* = *ab-near-semiring-one-zero* + *domain-near-semiring-one* +

assumes *dns06* [*simp*]: $d\ 0 = 0$

begin

lemma *domain-very-strict*: $d\ x = 0 \longleftrightarrow x = 0$

by (*metis annil dns1 dns06*)

lemma *dom-weakly-local*: $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$

proof –

have $x \cdot y = 0 \longleftrightarrow d\ (x \cdot y) = 0$

by (*simp add: domain-very-strict*)

also have $\dots \longleftrightarrow d\ (x \cdot d\ y) = 0$

by *simp*

finally show *?thesis*

using *domain-very-strict* **by** *blast*

qed

end

0.14.3 Domain Pre-Dioids

Pre-semirings with one and a left zero are automatically dioids. Hence there is no point defining domain pre-semirings separately from domain dioids. The axioms are once again from [?].

class *domain-pre-dioid-one* = *pre-dioid-one* + *domain-op* +

assumes *dvd1* : $x \leq d\ x \cdot x$

and *dvd2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$

and *dvd3* [*simp*]: $d\ x \leq 1$

and *dvd4* [*simp*]: $d\ (x + y) = d\ x + d\ y$

begin

We prepare to show that every domain pre-dioid with one is a domain near-dioid with one.

lemma *dns1''* [*simp*]: $d\ x \cdot x = x$

proof (*rule antisym*)

show $d\ x \cdot x \leq x$

using *dvd3 mult-isor* **by** *fastforce*

show $x \leq d\ x \cdot x$

by (*simp add: dvd1*)

qed

```

lemma d-iso:  $x \leq y \implies d\ x \leq d\ y$ 
  by (metis dpd4 less-eq-def)

lemma domain-1'':  $d\ (x \cdot y) \leq d\ x$ 
proof -
  have  $d\ (x \cdot y) = d\ (x \cdot d\ y)$ 
    by simp
  also have  $\dots \leq d\ (x \cdot 1)$ 
    by (meson d-iso dpd3 mult-isol)
  finally show ?thesis
    by simp
qed

lemma domain-export'' [simp]:  $d\ (d\ x \cdot y) = d\ x \cdot d\ y$ 
proof (rule antisym)
  have one:  $d\ (d\ x \cdot y) \leq d\ x$ 
    by (metis dpd2 domain-1'' mult-onel)
  have two:  $d\ (d\ x \cdot y) \leq d\ y$ 
    using d-iso dpd3 mult-isol by fastforce
  have  $d\ (d\ x \cdot y) = d\ (d\ (d\ x \cdot y)) \cdot d\ (d\ x \cdot y)$ 
    by simp
  also have  $\dots = d\ (d\ x \cdot y) \cdot d\ (d\ x \cdot y)$ 
    by (metis dns1'' dpd2 mult-assoc)
  thus  $d\ (d\ x \cdot y) \leq d\ x \cdot d\ y$ 
    using mult-isol-var one two by force
next
  have  $d\ x \cdot d\ y \leq 1$ 
    by (metis dpd3 mult-1-right mult-isol order.trans)
  thus  $d\ x \cdot d\ y \leq d\ (d\ x \cdot y)$ 
    by (metis dns1'' dpd2 mult-isol mult-oner)
qed

lemma dom-subid-aux1'':  $d\ x \cdot y \leq y$ 
proof -
  have  $d\ x \cdot y \leq 1 \cdot y$ 
    using dpd3 mult-isol by blast
  thus ?thesis
    by simp
qed

lemma dom-subid-aux2'':  $x \cdot d\ y \leq x$ 
  using dpd3 mult-isol by fastforce

lemma d-comm:  $d\ x \cdot d\ y = d\ y \cdot d\ x$ 
proof (rule antisym)
  have  $d\ x \cdot d\ y = (d\ x \cdot d\ y) \cdot (d\ x \cdot d\ y)$ 
    by (metis dns1'' domain-export'')
  thus  $d\ x \cdot d\ y \leq d\ y \cdot d\ x$ 

```

```

    by (metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var)
next
  have  $d\ y \cdot d\ x = (d\ y \cdot d\ x) \cdot (d\ y \cdot d\ x)$ 
  by (metis dns1'' domain-export'')
  thus  $d\ y \cdot d\ x \leq d\ x \cdot d\ y$ 
  by (metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var)
qed

subclass domain-near-semiring-one
  by (unfold-locales, auto simp: d-comm local.join.sup.absorb2)

lemma domain-subid:  $x \leq 1 \implies x \leq d\ x$ 
  by (metis dns1 mult-isol mult-oner)

lemma d-preserves-equation:  $d\ y \cdot x \leq x \cdot d\ z \longleftrightarrow d\ y \cdot x = d\ y \cdot x \cdot d\ z$ 
  by (metis dom-subid-aux2'' local.antisym local.dom-el-idem local.dom-subid-aux2
    local.order-prop local.subdistl mult-assoc)

lemma d-restrict-iff:  $(x \leq y) \longleftrightarrow (x \leq d\ x \cdot y)$ 
  by (metis dom-subid-aux2 dsg1 less-eq-def order-trans subdistl)

lemma d-restrict-iff-1:  $(d\ x \cdot y \leq z) \longleftrightarrow (d\ x \cdot y \leq d\ x \cdot z)$ 
  by (metis dom-subid-aux2 domain-1'' domain-invol dsg1 mult-isol-var order-trans)

end

We add once more a left unit of multiplication.

class domain-pre-dioid-one-zero1 = domain-pre-dioid-one + pre-dioid-one-zero1 +
  assumes dpd5 [simp]:  $d\ 0 = 0$ 

begin

subclass domain-near-semiring-one-zero1
  by (unfold-locales, simp)

end

```

0.14.4 Domain Semirings

We do not consider domain semirings without units separately at the moment. The axioms are taken from from [?]

```

class domain-semiring1 = semiring-one-zero1 + plus-ord + domain-op +
  assumes dsr1 [simp]:  $x + d\ x \cdot x = d\ x \cdot x$ 
  and dsr2 [simp]:  $d\ (x \cdot d\ y) = d\ (x \cdot y)$ 
  and dsr3 [simp]:  $d\ x + 1 = 1$ 
  and dsr4 [simp]:  $d\ 0 = 0$ 
  and dsr5 [simp]:  $d\ (x + y) = d\ x + d\ y$ 

```

begin

Every domain semiring is automatically a domain pre-dioid with one and left zero.

subclass *dioid-one-zero*
by (*standard*, *metis add-commute dsr1 dsr3 distrib-left mult-oner*)

subclass *domain-pre-dioid-one-zero*
by (*standard*, *auto simp: less-eq-def*)

end

class *domain-semiring* = *domain-semiringl* + *semiring-one-zero*

0.14.5 The Algebra of Domain Elements

We show that the domain elements of a domain semiring form a distributive lattice. Unfortunately we cannot prove this within the type class of domain semirings.

typedef (**overloaded**) *'a d-element* = $\{x :: 'a :: \text{domain-semiring}. x = d\ x\}$
by (*rule-tac x = 1 in exI*, *simp add: domain-subid order-class.eq-iff*)

setup-lifting *type-definition-d-element*

instantiation *d-element* :: (*domain-semiring*) *bounded-lattice*

begin

lift-definition *less-eq-d-element* :: *'a d-element* \Rightarrow *'a d-element* \Rightarrow *bool* **is** (\leq) .

lift-definition *less-d-element* :: *'a d-element* \Rightarrow *'a d-element* \Rightarrow *bool* **is** ($<$) .

lift-definition *bot-d-element* :: *'a d-element* **is** 0
by *simp*

lift-definition *top-d-element* :: *'a d-element* **is** 1
by *simp*

lift-definition *inf-d-element* :: *'a d-element* \Rightarrow *'a d-element* \Rightarrow *'a d-element* **is** (\cdot)
by (*metis dsg3*)

lift-definition *sup-d-element* :: *'a d-element* \Rightarrow *'a d-element* \Rightarrow *'a d-element* **is** ($+$)
by *simp*

instance

apply (*standard*; *transfer*)

apply (*simp add: less-le-not-le*)+


```

apply (metis dom-subid-aux2'')
apply (metis dom-subid-aux2)
apply (metis dom-glb)
apply simp+
by (metis dom-subid)

end

instance d-element :: (domain-semiring) distrib-lattice
  by (standard, transfer, metis dom-distrib)

```

0.14.6 Domain Semirings with a Greatest Element

If there is a greatest element in the semiring, then we have another equality.

class domain-semiring-top = domain-semiring + order-top

begin

notation top (\top)

lemma kat-equivalence-greatest: $d\ x \leq d\ y \longleftrightarrow x \leq d\ y \cdot \top$

proof

assume $d\ x \leq d\ y$

thus $x \leq d\ y \cdot \top$

by (metis dsq1 mult-isol-var top-greatest)

next

assume $x \leq d\ y \cdot \top$

thus $d\ x \leq d\ y$

using dom-glb-eq dom-iso **by** fastforce

qed

end

0.14.7 Forward Diamond Operators

context domain-semiringl

begin

We define a forward diamond operator over a domain semiring. A more modular consideration is not given at the moment.

definition fd :: 'a \Rightarrow 'a \Rightarrow 'a ((\mid -) -) [61,81] 82) **where**
 $\mid x \rangle y = d\ (x \cdot y)$

lemma fdia-d-simp [simp]: $\mid x \rangle d\ y = \mid x \rangle y$
by (simp add: fd-def)

lemma fdia-dom [simp]: $\mid x \rangle 1 = d\ x$
by (simp add: fd-def)

lemma *fdia-add1*: $|x\rangle (y + z) = |x\rangle y + |x\rangle z$
by (*simp add: fd-def distrib-left*)

lemma *fdia-add2*: $|x + y\rangle z = |x\rangle z + |y\rangle z$
by (*simp add: fd-def distrib-right*)

lemma *fdia-mult*: $|x \cdot y\rangle z = |x\rangle |y\rangle z$
by (*simp add: fd-def mult-assoc*)

lemma *fdia-one* [*simp*]: $|1\rangle x = d x$
by (*simp add: fd-def*)

lemma *fdemodalisation1*: $d z \cdot |x\rangle y = 0 \longleftrightarrow d z \cdot x \cdot d y = 0$
proof –
have $d z \cdot |x\rangle y = 0 \longleftrightarrow d z \cdot d (x \cdot y) = 0$
by (*simp add: fd-def*)
also have $\dots \longleftrightarrow d z \cdot x \cdot y = 0$
by (*metis annil dnso6 dsq1 dsq3 mult-assoc*)
finally show *?thesis*
using *dom-weakly-local* **by** *auto*
qed

lemma *fdemodalisation2*: $|x\rangle y \leq d z \longleftrightarrow x \cdot d y \leq d z \cdot x$
proof
assume $|x\rangle y \leq d z$
hence $a: d (x \cdot d y) \leq d z$
by (*simp add: fd-def*)
have $x \cdot d y = d (x \cdot d y) \cdot x \cdot d y$
using *dsq1 mult-assoc* **by** *presburger*
also have $\dots \leq d z \cdot x \cdot d y$
using *a calculation dom-llp2 mult-assoc* **by** *auto*
finally show $x \cdot d y \leq d z \cdot x$
using *dom-subid-aux2'' order-trans* **by** *blast*
next
assume $x \cdot d y \leq d z \cdot x$
hence $d (x \cdot d y) \leq d (d z \cdot d x)$
using *dom-iso* **by** *fastforce*
also have $\dots \leq d (d z)$
using *domain-1''* **by** *blast*
finally show $|x\rangle y \leq d z$
by (*simp add: fd-def*)
qed

lemma *fd-iso1*: $d x \leq d y \implies |z\rangle x \leq |z\rangle y$
using *fd-def local.dom-iso local.mult-isol* **by** *fastforce*

lemma *fd-iso2*: $x \leq y \implies |x\rangle z \leq |y\rangle z$
by (*simp add: fd-def dom-iso mult-isor*)

```

lemma fd-zero-var [simp]:  $|0\rangle x = 0$ 
  by (simp add: fd-def)

lemma fd-subdist-1:  $|x\rangle y \leq |x\rangle (y + z)$ 
  by (simp add: fd-iso1)

lemma fd-subdist-2:  $|x\rangle (d y \cdot d z) \leq |x\rangle y$ 
  by (simp add: fd-iso1 dom-subid-aux2'')

lemma fd-subdist:  $|x\rangle (d y \cdot d z) \leq |x\rangle y \cdot |x\rangle z$ 
  using fd-def fd-iso1 fd-subdist-2 dom-glb dom-subid-aux2 by auto

lemma fdia-export-1:  $d y \cdot |x\rangle z = |d y \cdot x\rangle z$ 
  by (simp add: fd-def mult-assoc)

```

end

context *domain-semiring*

begin

```

lemma fdia-zero [simp]:  $|x\rangle 0 = 0$ 
  by (simp add: fd-def)

```

end

0.14.8 Domain Kleene Algebras

We add the Kleene star to our considerations. Special domain axioms are not needed.

class *domain-left-kleene-algebra* = *left-kleene-algebra-zero1* + *domain-semiring1*

begin

```

lemma dom-star [simp]:  $d (x^*) = 1$ 
proof –
  have  $d (x^*) = d (1 + x \cdot x^*)$ 
    by simp
  also have  $\dots = d 1 + d (x \cdot x^*)$ 
    using dns3 by blast
  finally show ?thesis
    using add-commute local.dsr3 by auto
qed

```

```

lemma fdia-star-unfold [simp]:  $|1\rangle y + |x\rangle |x^*\rangle y = |x^*\rangle y$ 
proof –
  have  $|1\rangle y + |x\rangle |x^*\rangle y = |1 + x \cdot x^*\rangle y$ 
    using local.fdia-add2 local.fdia-mult by presburger

```

thus ?thesis
 by simp
 qed

lemma *fdia-star-unfoldr* [simp]: $|1\rangle y + |x^*\rangle |x\rangle y = |x^*\rangle y$
proof –
 have $|1\rangle y + |x^*\rangle |x\rangle y = |1 + x^* \cdot x\rangle y$
 using *fdia-add2 fdia-mult* **by** *presburger*
 thus ?thesis
 by simp
 qed

lemma *fdia-star-unfold-var* [simp]: $d y + |x\rangle |x^*\rangle y = |x^*\rangle y$
proof –
 have $d y + |x\rangle |x^*\rangle y = |1\rangle y + |x\rangle |x^*\rangle y$
 by simp
 also have $\dots = |1 + x \cdot x^*\rangle y$
 using *fdia-add2 fdia-mult* **by** *presburger*
 finally **show** ?thesis
 by simp
 qed

lemma *fdia-star-unfoldr-var* [simp]: $d y + |x^*\rangle |x\rangle y = |x^*\rangle y$
proof –
 have $d y + |x^*\rangle |x\rangle y = |1\rangle y + |x^*\rangle |x\rangle y$
 by simp
 also have $\dots = |1 + x^* \cdot x\rangle y$
 using *fdia-add2 fdia-mult* **by** *presburger*
 finally **show** ?thesis
 by simp
 qed

lemma *fdia-star-induct-var*: $|x\rangle y \leq d y \implies |x^*\rangle y \leq d y$
proof –
 assume *a1*: $|x\rangle y \leq d y$
 hence $x \cdot d y \leq d y \cdot x$
 by (*simp add: fdemodalisation2*)
 hence $x^* \cdot d y \leq d y \cdot x^*$
 by (*simp add: star-sim1*)
 thus ?thesis
 by (*simp add: fdemodalisation2*)
 qed

lemma *fdia-star-induct*: $d z + |x\rangle y \leq d y \implies |x^*\rangle z \leq d y$
proof –
 assume *a*: $d z + |x\rangle y \leq d y$
 hence *b*: $d z \leq d y$ **and** *c*: $|x\rangle y \leq d y$
 apply (*simp add: local.join.le-supE*)
 using *a* **by** *auto*

```

  hence  $d: |x^*\rangle z \leq |x^*\rangle y$ 
    using fd-def fd-iso1 by auto
  have  $|x^*\rangle y \leq d y$ 
    using c fdia-star-induct-var by blast
  thus ?thesis
    using d by fastforce
qed

lemma fdia-star-induct-eq:  $d z + |x\rangle y = d y \implies |x^*\rangle z \leq d y$ 
  by (simp add: fdia-star-induct)

end

class domain-kleene-algebra = kleene-algebra + domain-semiring

begin

subclass domain-left-kleene-algebra ..

end

end

```

0.15 Antidomain Semirings

```

theory Antidomain-Semiring
imports Domain-Semiring
begin

```

0.15.1 Antidomain Monoids

We axiomatise antidomain monoids, using the axioms of [?].

```

class antidomain-op =
  fixes antidomain-op :: 'a  $\Rightarrow$  'a (ad)

class antidomain-left-monoid = monoid-mult + antidomain-op +
  assumes am1 [simp]:  $ad\ x \cdot x = ad\ 1$ 
  and am2:  $ad\ x \cdot ad\ y = ad\ y \cdot ad\ x$ 
  and am3 [simp]:  $ad\ (ad\ x) \cdot x = x$ 
  and am4 [simp]:  $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$ 
  and am5 [simp]:  $ad\ (x \cdot y) \cdot x \cdot ad\ y = ad\ (x \cdot y) \cdot x$ 

begin

no-notation domain-op (d)
no-notation zero-class.zero (0)

```

We define a zero element and operations of domain and addition.

definition $a\text{-zero} :: 'a \Rightarrow (0)$ **where**
 $0 = ad\ 1$

definition $am\text{-}d :: 'a \Rightarrow 'a \Rightarrow (d)$ **where**
 $d\ x = ad\ (ad\ x)$

definition $am\text{-}add\text{-}op :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** \oplus 65) **where**
 $x \oplus y \equiv ad\ (ad\ x \cdot ad\ y)$

lemma $a\text{-}d\text{-}zero$ [*simp*]: $ad\ x \cdot d\ x = 0$
by (*metis* $am1\ am2\ a\text{-}zero\text{-}def\ am\text{-}d\text{-}def$)

lemma $a\text{-}d\text{-}one$ [*simp*]: $d\ x \oplus ad\ x = 1$
by (*metis* $am1\ am3\ mult\text{-}1\text{-}right\ am\text{-}d\text{-}def\ am\text{-}add\text{-}op\text{-}def$)

lemma $n\text{-}annil$ [*simp*]: $0 \cdot x = 0$
proof –
have $0 \cdot x = d\ x \cdot ad\ x \cdot x$
by (*simp* *add*: $a\text{-}zero\text{-}def\ am\text{-}d\text{-}def$)
also have $\dots = d\ x \cdot 0$
by (*metis* $am1\ mult\text{-}assoc\ a\text{-}zero\text{-}def$)
thus *?thesis*
by (*metis* $am1\ am2\ am3\ mult\text{-}assoc\ a\text{-}zero\text{-}def$)
qed

lemma $a\text{-}mult\text{-}idem$ [*simp*]: $ad\ x \cdot ad\ x = ad\ x$
proof –
have $ad\ x \cdot ad\ x = ad\ (1 \cdot x) \cdot 1 \cdot ad\ x$
by *simp*
also have $\dots = ad\ (1 \cdot x) \cdot 1$
using $am5$ **by** *blast*
finally show *?thesis*
by *simp*
qed

lemma $a\text{-}add\text{-}idem$ [*simp*]: $ad\ x \oplus ad\ x = ad\ x$
by (*metis* $am1\ am3\ am4\ mult\text{-}1\text{-}right\ am\text{-}add\text{-}op\text{-}def$)

The next three axioms suffice to show that the domain elements form a Boolean algebra.

lemma $a\text{-}add\text{-}comm$: $x \oplus y = y \oplus x$
using $am2\ am\text{-}add\text{-}op\text{-}def$ **by** *auto*

lemma $a\text{-}add\text{-}assoc$: $x \oplus (y \oplus z) = (x \oplus y) \oplus z$
proof –
have $\bigwedge x\ y. ad\ x \cdot ad\ (x \cdot y) = ad\ x$
by (*metis* $a\text{-}mult\text{-}idem\ am2\ am4\ mult\text{-}assoc$)
thus *?thesis*
by (*metis* $a\text{-}add\text{-}comm\ am\text{-}add\text{-}op\text{-}def\ local.am3\ local.am4\ mult\text{-}assoc$)

qed

lemma *huntington* [*simp*]: $ad\ (x \oplus y) \oplus ad\ (x \oplus ad\ y) = ad\ x$
using *a-add-idem am-add-op-def* **by** *auto*

lemma *a-absorb1* [*simp*]: $(ad\ x \oplus ad\ y) \cdot ad\ x = ad\ x$
by (*metis a-add-idem a-mult-idem am4 mult-assoc am-add-op-def*)

lemma *a-absorb2* [*simp*]: $ad\ x \oplus ad\ x \cdot ad\ y = ad\ x$
proof –
have $ad\ (ad\ x) \cdot ad\ (ad\ x \cdot ad\ y) = ad\ (ad\ x)$
by (*metis (no-types) a-mult-idem local.am4 local.mult.semigroup-axioms semigroup.assoc*)
then show *?thesis*
using *a-add-idem am-add-op-def* **by** *auto*
 qed

The distributivity laws remain to be proved; our proofs follow those of Mad-
 dux [?].

lemma *prod-split* [*simp*]: $ad\ x \cdot ad\ y \oplus ad\ x \cdot d\ y = ad\ x$
using *a-add-idem am-d-def am-add-op-def* **by** *auto*

lemma *sum-split* [*simp*]: $(ad\ x \oplus ad\ y) \cdot (ad\ x \oplus d\ y) = ad\ x$
using *a-add-idem am-d-def am-add-op-def* **by** *fastforce*

lemma *a-comp-simp* [*simp*]: $(ad\ x \oplus ad\ y) \cdot d\ x = ad\ y \cdot d\ x$
proof –
have $f1: (ad\ x \oplus ad\ y) \cdot d\ x = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ (ad\ x) \cdot ad\ (ad\ (ad\ y))$
by (*simp add: am-add-op-def am-d-def*)
have $f2: ad\ y = ad\ (ad\ (ad\ y))$
using *a-add-idem am-add-op-def* **by** *auto*
have $ad\ y = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ y$
by (*metis (no-types) a-absorb1 a-add-comm am-add-op-def*)
then show *?thesis*
using $f2\ f1$ **by** (*simp add: am-d-def local.am2 local.mult.semigroup-axioms semigroup.assoc*)
 qed

lemma *a-distrib1*: $ad\ x \cdot (ad\ y \oplus ad\ z) = ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ z$
proof –
have $f1: \bigwedge a. ad\ (ad\ (ad\ (a::'a)) \cdot ad\ (ad\ a)) = ad\ a$
using *a-add-idem am-add-op-def* **by** *auto*
have $f2: \bigwedge a\ aa. ad\ ((a::'a) \cdot aa) \cdot (a \cdot ad\ aa) = ad\ (a \cdot aa) \cdot a$
using *local.am5 mult-assoc* **by** *auto*
have $f3: \bigwedge a. ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using $f1$ **by** *simp*
have $\bigwedge a. ad\ (a::'a) \cdot ad\ a = ad\ a$
by *simp*

then have $\bigwedge a \text{ aa. } ad (ad (ad (a::'a) \cdot ad \text{ aa})) = ad \text{ aa} \cdot ad \text{ a}$
 using *f3 f2* by (*metis (no-types) local.am2 local.am4 mult-assoc*)
 then have $ad \text{ x} \cdot (ad \text{ y} \oplus ad \text{ z}) = ad \text{ x} \cdot (ad \text{ y} \oplus ad \text{ z}) \cdot ad \text{ y} \oplus ad \text{ x} \cdot (ad \text{ y}$
 $\oplus ad \text{ z}) \cdot d \text{ y}$
 using *am-add-op-def am-d-def local.am2 local.am4* by *presburger*
 also have $\dots = ad \text{ x} \cdot ad \text{ y} \oplus ad \text{ x} \cdot (ad \text{ y} \oplus ad \text{ z}) \cdot d \text{ y}$
 by (*simp add: mult-assoc*)
 also have $\dots = ad \text{ x} \cdot ad \text{ y} \oplus ad \text{ x} \cdot ad \text{ z} \cdot d \text{ y}$
 by (*simp add: mult-assoc*)
 also have $\dots = ad \text{ x} \cdot ad \text{ y} \oplus ad \text{ x} \cdot ad \text{ y} \cdot ad \text{ z} \oplus ad \text{ x} \cdot ad \text{ z} \cdot d \text{ y}$
 by (*metis a-add-idem a-mult-idem local.am4 mult-assoc am-add-op-def*)
 also have $\dots = ad \text{ x} \cdot ad \text{ y} \oplus (ad \text{ x} \cdot ad \text{ z} \cdot ad \text{ y} \oplus ad \text{ x} \cdot ad \text{ z} \cdot d \text{ y})$
 by (*metis am2 mult-assoc a-add-assoc*)
 finally show *?thesis*
 by (*metis a-add-idem a-mult-idem am4 am-d-def am-add-op-def*)
 qed

lemma *a-distrib2*: $ad \text{ x} \oplus ad \text{ y} \cdot ad \text{ z} = (ad \text{ x} \oplus ad \text{ y}) \cdot (ad \text{ x} \oplus ad \text{ z})$
proof –
 have *f1*: $\bigwedge a \text{ aa ab. } ad (ad (ad (a::'a) \cdot ad \text{ aa}) \cdot ad (ad \text{ a} \cdot ad \text{ ab})) = ad \text{ a} \cdot ad$
 $(ad (ad \text{ aa}) \cdot ad (ad \text{ ab}))$
 using *a-distrib1 am-add-op-def* by *auto*
 have $\bigwedge a. ad (ad (ad (a::'a))) = ad \text{ a}$
 by (*metis a-absorb2 a-mult-idem am-add-op-def*)
 then have $ad (ad (ad \text{ x}) \cdot ad (ad \text{ y})) \cdot ad (ad (ad \text{ x}) \cdot ad (ad \text{ z})) = ad (ad (ad$
 $\text{x}) \cdot ad (ad \text{ y} \cdot ad \text{ z}))$
 using *f1* by (*metis (full-types)*)
 then show *?thesis*
 by (*simp add: am-add-op-def*)
 qed

lemma *aa-loc [simp]*: $d (x \cdot d \text{ y}) = d (x \cdot y)$
proof –
 have *f1*: $x \cdot d \text{ y} \cdot y = x \cdot y$
 by (*metis am3 mult-assoc am-d-def*)
 have *f2*: $\bigwedge w \text{ z. } ad (w \cdot z) \cdot (w \cdot ad \text{ z}) = ad (w \cdot z) \cdot w$
 by (*metis am5 mult-assoc*)
 hence *f3*: $\bigwedge z. ad (x \cdot y) \cdot (x \cdot z) = ad (x \cdot y) \cdot (x \cdot (ad (ad (ad \text{ y}) \cdot y) \cdot z))$
 using *f1* by (*metis (no-types) mult-assoc am-d-def*)
 have $ad (x \cdot ad (ad \text{ y})) \cdot (x \cdot y) = 0$ using *f1*
 by (*metis am1 mult-assoc n-annil a-zero-def am-d-def*)
 thus *?thesis*
 by (*metis a-d-zero am-d-def f3 local.am1 local.am2 local.am3 local.am4*)
 qed

lemma *a-loc [simp]*: $ad (x \cdot d \text{ y}) = ad (x \cdot y)$
proof –
 have $\bigwedge a. ad (ad (ad (a::'a))) = ad \text{ a}$
 using *am-add-op-def am-d-def prod-split* by *auto*


```

then show ?thesis
  by (metis (full-types) aa-loc am-d-def)
qed

lemma d-a-export [simp]:  $d (ad\ x \cdot y) = ad\ x \cdot d\ y$ 
proof –
  have f1:  $\bigwedge a\ aa. ad\ ((a::'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$ 
    using a-loc am-d-def by auto
  have  $\bigwedge a. ad\ (ad\ (a::'a) \cdot a) = 1$ 
    using a-d-one am-add-op-def am-d-def by auto
  then have  $\bigwedge a\ aa. ad\ (ad\ (ad\ (a::'a) \cdot ad\ aa)) = ad\ a \cdot ad\ aa$ 
    using f1 by (metis a-distrib2 am-add-op-def local.mult-1-left)
  then show ?thesis
    using f1 by (metis (no-types) am-d-def)
qed

```

Every antidomain monoid is a domain monoid.

```

sublocale dm: domain-monoid am-d ( $\cdot$ ) 1
  apply (unfold-locales)
  apply (simp add: am-d-def)
  apply simp
  using am-d-def d-a-export apply auto[1]
  by (simp add: am-d-def local.am2)

```

```

lemma ds-ord-iso1:  $x \sqsubseteq y \implies z \cdot x \sqsubseteq z \cdot y$ 

```

oops

```

lemma a-very-costrict:  $ad\ x = 1 \longleftrightarrow x = 0$ 
proof
  assume a:  $ad\ x = 1$ 
  hence  $0 = ad\ x \cdot x$ 
    using a-zero-def by force
  thus  $x = 0$ 
    by (simp add: a)
next
  assume  $x = 0$ 
  thus  $ad\ x = 1$ 
    using a-zero-def am-d-def dm.dom-one by auto
qed

```

```

lemma a-weak-loc:  $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$ 
proof –
  have  $x \cdot y = 0 \longleftrightarrow ad\ (x \cdot y) = 1$ 
    by (simp add: a-very-costrict)
  also have  $\dots \longleftrightarrow ad\ (x \cdot d\ y) = 1$ 
    by simp
  finally show ?thesis
    using a-very-costrict by blast

```

qed

lemma *a-closure* [simp]: $d (ad\ x) = ad\ x$
using *a-add-idem am-add-op-def am-d-def* **by** *auto*

lemma *a-d-mult-closure* [simp]: $d (ad\ x \cdot ad\ y) = ad\ x \cdot ad\ y$
by *simp*

lemma *kat-3'*: $d\ x \cdot y \cdot ad\ z = 0 \implies d\ x \cdot y = d\ x \cdot y \cdot d\ z$
by (*metis dm.dom-one local.am5 local.mult-1-left a-zero-def am-d-def*)

lemma *s4* [simp]: $ad\ x \cdot ad (ad\ x \cdot y) = ad\ x \cdot ad\ y$
proof –
have $\bigwedge a\ aa. ad (a::'a) \cdot ad (ad\ aa) = ad (ad (ad\ a \cdot aa))$
using *am-d-def d-a-export* **by** *presburger*
then have $\bigwedge a\ aa. ad (ad (a::'a)) \cdot ad\ aa = ad (ad (ad\ aa \cdot a))$
using *local.am2* **by** *presburger*
then show *?thesis*
by (*metis a-comp-simp a-d-mult-closure am-add-op-def am-d-def local.am2*)
qed

end

class *antidomain-monoid* = *antidomain-left-monoid* +
assumes *am6* [simp]: $x \cdot ad\ 1 = ad\ 1$

begin

lemma *kat-3-equiv*: $d\ x \cdot y \cdot ad\ z = 0 \iff d\ x \cdot y = d\ x \cdot y \cdot d\ z$
apply *standard*
apply (*metis kat-3'*)
by (*simp add: mult-assoc a-zero-def am-d-def*)

no-notation *a-zero* (*0*)

no-notation *am-d* (*d*)

end

0.15.2 Antidomain Near-Semirings

We define antidomain near-semirings. We do not consider units separately. The axioms are taken from [?].

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

class *antidomain-near-semiring* = *ab-near-semiring-one-zero1* + *antidomain-op* +
plus-ord +
assumes *ans1* [simp]: $ad\ x \cdot x = 0$
and *ans2* [simp]: $ad (x \cdot y) + ad (x \cdot ad (ad\ y)) = ad (x \cdot ad (ad\ y))$
and *ans3* [simp]: $ad (ad\ x) + ad\ x = 1$

```

and ans4 [simp]:  $ad\ (x + y) = ad\ x \cdot ad\ y$ 

begin

definition ans-d :: 'a  $\Rightarrow$  'a (d) where
   $d\ x = ad\ (ad\ x)$ 

lemma a-a-one [simp]:  $d\ 1 = 1$ 
proof -
  have  $d\ 1 = d\ 1 + 0$ 
  by simp
  also have  $\dots = d\ 1 + ad\ 1$ 
  by (metis ans1 mult-1-right)
  finally show ?thesis
  by (simp add: ans-d-def)
qed

lemma a-very-costrict':  $ad\ x = 1 \longleftrightarrow x = 0$ 
proof
  assume  $ad\ x = 1$ 
  hence  $x = ad\ x \cdot x$ 
  by simp
  thus  $x = 0$ 
  by auto
next
  assume  $x = 0$ 
  hence  $ad\ x = ad\ 0$ 
  by blast
  thus  $ad\ x = 1$ 
  by (metis a-a-one ans-d-def local.ans1 local.mult-1-right)
qed

lemma one-idem [simp]:  $1 + 1 = 1$ 
proof -
  have  $1 + 1 = d\ 1 + d\ 1$ 
  by simp
  also have  $\dots = ad\ (ad\ 1 \cdot 1) + ad\ (ad\ 1 \cdot d\ 1)$ 
  using a-a-one ans-d-def by auto
  also have  $\dots = ad\ (ad\ 1 \cdot d\ 1)$ 
  using ans-d-def local.ans2 by presburger
  also have  $\dots = ad\ (ad\ 1 \cdot 1)$ 
  by simp
  also have  $\dots = d\ 1$ 
  by (simp add: ans-d-def)
  finally show ?thesis
  by simp
qed

```

Every antidomain near-semiring is automatically a dioid, and therefore or-

dered.

```

subclass near-diooid-one-zero1
proof
  show  $\bigwedge x. x + x = x$ 
  proof -
    fix x
    have  $x + x = 1 \cdot x + 1 \cdot x$ 
      by simp
    also have  $\dots = (1 + 1) \cdot x$ 
      using distrib-right' by presburger
    finally show  $x + x = x$ 
      by simp
  qed
qed

```

```

lemma d1-a [simp]:  $d\ x \cdot x = x$ 
proof -
  have  $x = (d\ x + ad\ x) \cdot x$ 
    by (simp add: ans-d-def)
  also have  $\dots = d\ x \cdot x + ad\ x \cdot x$ 
    using distrib-right' by blast
  also have  $\dots = d\ x \cdot x + 0$ 
    by simp
  finally show ?thesis
    by auto
qed

```

```

lemma a-comm:  $ad\ x \cdot ad\ y = ad\ y \cdot ad\ x$ 
  using add-commute ans4 by fastforce

```

```

lemma a-subid:  $ad\ x \leq 1$ 
  using local.ans3 local.join.sup-ge2 by fastforce

```

```

lemma a-subid-aux1:  $ad\ x \cdot y \leq y$ 
  using a-subid mult-isor by fastforce

```

```

lemma a-subdist:  $ad\ (x + y) \leq ad\ x$ 
  by (metis a-subid-aux1 ans4 add-comm)

```

```

lemma a-antitone:  $x \leq y \implies ad\ y \leq ad\ x$ 
  using a-subdist local.order-prop by auto

```

```

lemma a-mul-d [simp]:  $ad\ x \cdot d\ x = 0$ 
  by (metis a-comm ans-d-def local.ans1)

```

```

lemma a-gla1:  $ad\ x \cdot y = 0 \implies ad\ x \leq ad\ y$ 
proof -
  assume  $ad\ x \cdot y = 0$ 
  hence a:  $ad\ x \cdot d\ y = 0$ 

```

```

  by (metis a-subid a-very-costrict' ans-d-def local.ans2 local.join.sup.order-iff)
have ad x = (d y + ad y) · ad x
  by (simp add: ans-d-def)
also have ... = d y · ad x + ad y · ad x
  using distrib-right' by blast
also have ... = ad x · d y + ad x · ad y
  using a-comm ans-d-def by auto
also have ... = ad x · ad y
  by (simp add: a)
finally show ad x ≤ ad y
  by (metis a-subid-aux1)
qed

```

```

lemma a-gla2: ad x ≤ ad y ⇒ ad x · y = 0
proof -
  assume ad x ≤ ad y
  hence ad x · y ≤ ad y · y
    using mult-isor by blast
  thus ?thesis
    by (simp add: join.le-bot)
qed

```

```

lemma a2-eq [simp]: ad (x · d y) = ad (x · y)
proof (rule antisym)
  show ad (x · y) ≤ ad (x · d y)
    by (simp add: ans-d-def local.less-eq-def)
next
  show ad (x · d y) ≤ ad (x · y)
    by (metis a-gla1 a-mul-d ans1 d1-a mult-assoc)
qed

```

```

lemma a-export' [simp]: ad (ad x · y) = d x + ad y
proof (rule antisym)
  have ad (ad x · y) · ad x · d y = 0
    by (simp add: a-gla2 local.mult.semigroup-axioms semigroup.assoc)
  hence a: ad (ad x · y) · d y ≤ ad (ad x)
    by (metis a-comm a-gla1 ans4 mult-assoc ans-d-def)
  have ad (ad x · y) = ad (ad x · y) · d y + ad (ad x · y) · ad y
    by (metis (no-types) add-commute ans3 ans4 distrib-right' mult-onel ans-d-def)
  thus ad (ad x · y) ≤ d x + ad y
    by (metis a-subid-aux1 a-join.sup-mono ans-d-def)
next
  show d x + ad y ≤ ad (ad x · y)
    by (metis a2-eq a-antitone a-comm a-subid-aux1 join.sup-least ans-d-def)
qed

```

Every antidomain near-semiring is a domain near-semiring.

```

sublocale dnsz: domain-near-semiring-one-zero1 (+) (·) 1 0 ans-d (≤) (<)
  apply (unfold-locales)

```

```

apply simp
using a2-eq ans-d-def apply auto[1]
apply (simp add: a-subid ans-d-def local.join.sup-absorb2)
apply (simp add: ans-d-def)
apply (simp add: a-comm ans-d-def)
using a-a-one a-very-costrict' ans-d-def by force

```

```

lemma a-idem [simp]: ad x · ad x = ad x
proof –
  have ad x = (d x + ad x) · ad x
    by (simp add: ans-d-def)
  also have ... = d x · ad x + ad x · ad x
    using distrib-right' by blast
  finally show ?thesis
    by (simp add: ans-d-def)
qed

```

```

lemma a-3-var [simp]: ad x · ad y · (x + y) = 0
  by (metis ans1 ans4)

```

```

lemma a-3 [simp]: ad x · ad y · d (x + y) = 0
  by (metis a-mul-d ans4)

```

```

lemma a-closure' [simp]: d (ad x) = ad x
proof –
  have d (ad x) = ad (1 · d x)
    by (simp add: ans-d-def)
  also have ... = ad (1 · x)
    using a2-eq by blast
  finally show ?thesis
    by simp
qed

```

The following counterexamples show that some of the antidomain monoid axioms do not need to hold.

```

lemma x · ad 1 = ad 1

```

oops

```

lemma ad (x · y) · ad (x · ad y) = ad x

```

oops

```

lemma ad (x · y) · ad (x · ad y) = ad x

```

oops

```

lemma phl-seq-inv: d v · x · y · ad w = 0 ⇒ ∃ z. d v · x · d z = 0 ∧ ad z · y · ad w = 0

```

proof –

assume $d \ v \cdot x \cdot y \cdot ad \ w = 0$
hence $d \ v \cdot x \cdot d \ (y \cdot ad \ w) = 0 \wedge ad \ (y \cdot ad \ w) \cdot y \cdot ad \ w = 0$
by (*metis dnsz.dom-weakly-local local.ans1 mult-assoc*)
thus $\exists z. d \ v \cdot x \cdot d \ z = 0 \wedge ad \ z \cdot y \cdot ad \ w = 0$
by *blast*

qed

lemma *a-fixpoint*: $ad \ x = x \implies (\forall y. y = 0)$

proof –

assume $a1: ad \ x = x$
{ fix $aa :: 'a$
have $aa = 0$
using $a1$ **by** (*metis (no-types) a-mul-d ans-d-def local.annil local.ans3 local.join.sup.idem local.mult-1-left*)
}
then show *?thesis*
by *blast*

qed

no-notation $ans-d \ (d)$

end

0.15.3 Antidomain Pre-Dioids

Antidomain pre-dioids are based on a different set of axioms, which are again taken from [?].

class *antidomain-pre-dioid* = *pre-dioid-one-zero* + *antidomain-op* +
assumes $apd1$ [*simp*]: $ad \ x \cdot x = 0$
and $apd2$ [*simp*]: $ad \ (x \cdot y) \leq ad \ (x \cdot ad \ (ad \ y))$
and $apd3$ [*simp*]: $ad \ (ad \ x) + ad \ x = 1$

begin

definition $apd-d :: 'a \Rightarrow 'a \ (d)$ **where**

$d \ x = ad \ (ad \ x)$

lemma *a-very-costrict''*: $ad \ x = 1 \longleftrightarrow x = 0$

by (*metis add-commute local.add-zero local.antisym local.apd1 local.apd3 local.join.bot-least local.mult-1-right local.phl-skip*)

lemma *a-subid'*: $ad \ x \leq 1$

using $local.apd3$ $local.join.sup-ge2$ **by** *fastforce*

lemma $d1-a'$ [*simp*]: $d \ x \cdot x = x$

proof –

have $x = (d \ x + ad \ x) \cdot x$
by (*simp add: apd-d-def*)

```

also have ... =  $d\ x \cdot x + ad\ x \cdot x$ 
  using distrib-right' by blast
also have ... =  $d\ x \cdot x + 0$ 
  by simp
finally show ?thesis
  by auto
qed

```

```

lemma a-subid-aux1':  $ad\ x \cdot y \leq y$ 
  using a-subid' mult-isol by fastforce

```

```

lemma a-mul-d' [simp]:  $ad\ x \cdot d\ x = 0$ 
proof –
  have  $1 = ad\ (ad\ x \cdot x)$ 
    using a-very-costrict'' by force
  thus ?thesis
    by (metis a-subid' a-very-costrict'' apd-d-def local.antisym local.apd2)
qed

```

```

lemma a-d-closed [simp]:  $d\ (ad\ x) = ad\ x$ 
proof (rule antisym)
  have  $d\ (ad\ x) = (d\ x + ad\ x) \cdot d\ (ad\ x)$ 
    by (simp add: apd-d-def)
  also have ... =  $ad\ (ad\ x) \cdot ad\ (d\ x) + ad\ x \cdot d\ (ad\ x)$ 
    using apd-d-def local.distrib-right' by presburger
  also have ... =  $ad\ x \cdot d\ (ad\ x)$ 
    using a-mul-d' apd-d-def by auto
  finally show  $d\ (ad\ x) \leq ad\ x$ 
    by (metis a-subid' mult-1-right mult-isol apd-d-def)
next
  have  $ad\ x = ad\ (1 \cdot x)$ 
    by simp
  also have ...  $\leq ad\ (1 \cdot d\ x)$ 
    using apd-d-def local.apd2 by presburger
  also have ... =  $ad\ (d\ x)$ 
    by simp
  finally show  $ad\ x \leq d\ (ad\ x)$ 
    by (simp add: apd-d-def)
qed

```

```

lemma meet-ord-def:  $ad\ x \leq ad\ y \longleftrightarrow ad\ x \cdot ad\ y = ad\ x$ 
  by (metis a-d-closed a-subid-aux1' d1-a' eq-iff mult-1-right mult-isol)

```

```

lemma d-weak-loc:  $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$ 
proof –
  have  $x \cdot y = 0 \longleftrightarrow ad\ (x \cdot y) = 1$ 
    by (simp add: a-very-costrict'')
  also have ...  $\longleftrightarrow ad\ (x \cdot d\ y) = 1$ 
    by (metis apd1 apd2 a-subid' apd-d-def d1-a' eq-iff mult-1-left mult-assoc)

```


finally show *?thesis*
by (*simp add: a-very-costrict''*)
qed

lemma gla-1: $ad\ x \cdot y = 0 \implies ad\ x \leq ad\ y$
proof –
assume $ad\ x \cdot y = 0$
hence $a: ad\ x \cdot d\ y = 0$
using *d-weak-loc* **by** *force*
hence $d\ y = ad\ x \cdot d\ y + d\ y$
by *simp*
also have $\dots = (1 + ad\ x) \cdot d\ y$
using *join.sup-commute* **by** *auto*
also have $\dots = (d\ x + ad\ x) \cdot d\ y$
using *apd-d-def calculation* **by** *auto*
also have $\dots = d\ x \cdot d\ y$
by (*simp add: a join.sup-commute*)
finally have $d\ y \leq d\ x$
by (*metis apd-d-def a-subid' mult-1-right mult-isol*)
hence $d\ y \cdot ad\ x = 0$
by (*metis apd-d-def a-d-closed a-mul-d' distrib-right' less-eq-def no-trivial-inverse*)
hence $ad\ x = ad\ y \cdot ad\ x$
by (*metis apd-d-def apd3 add-0-left distrib-right' mult-1-left*)
thus $ad\ x \leq ad\ y$
by (*metis add-commute apd3 mult-oner subdistl*)
qed

lemma a2-eq' [simp]: $ad\ (x \cdot d\ y) = ad\ (x \cdot y)$
proof (*rule antisym*)
show $ad\ (x \cdot y) \leq ad\ (x \cdot d\ y)$
by (*simp add: apd-d-def*)
next
show $ad\ (x \cdot d\ y) \leq ad\ (x \cdot y)$
by (*metis gla-1 apd1 a-mul-d' d1-a' mult-assoc*)
qed

lemma a-supdist-var: $ad\ (x + y) \leq ad\ x$
by (*metis gla-1 apd1 join.le-bot subdistl*)

lemma a-antitone': $x \leq y \implies ad\ y \leq ad\ x$
using *a-supdist-var local.order-prop* **by** *auto*

lemma a-comm-var: $ad\ x \cdot ad\ y \leq ad\ y \cdot ad\ x$
proof –
have $ad\ x \cdot ad\ y = d\ (ad\ x \cdot ad\ y) \cdot ad\ x \cdot ad\ y$
by (*simp add: mult-assoc*)
also have $\dots \leq d\ (ad\ x \cdot ad\ y) \cdot ad\ x$
using *a-subid' mult-isol* **by** *fastforce*
also have $\dots \leq d\ (ad\ y) \cdot ad\ x$

```

    by (simp add: a-antitone' a-subid-aux1' apd-d-def local.mult-isor)
  finally show ?thesis
    by simp
qed

```

```

lemma a-comm': ad x · ad y = ad y · ad x
  by (simp add: a-comm-var eq-iff)

```

```

lemma a-closed [simp]: d (ad x · ad y) = ad x · ad y
proof -
  have f1:  $\bigwedge x y. ad\ x \leq ad\ (ad\ y \cdot x)$ 
    by (simp add: a-antitone' a-subid-aux1')
  have  $\bigwedge x y. d\ (ad\ x \cdot y) \leq ad\ x$ 
    by (metis a2-eq' a-antitone' a-comm' a-d-closed apd-d-def f1)
  hence  $\bigwedge x y. d\ (ad\ x \cdot y) \cdot y = ad\ x \cdot y$ 
    by (metis d1-a' meet-ord-def mult-assoc apd-d-def)
  thus ?thesis
    by (metis f1 a-comm' apd-d-def meet-ord-def)
qed

```

```

lemma a-export'' [simp]: ad (ad x · y) = d x + ad y
proof (rule antisym)
  have ad (ad x · y) · ad x · d y = 0
    using d-weak-loc mult-assoc by fastforce
  hence a: ad (ad x · y) · d y ≤ d x
    by (metis a-closed a-comm' apd-d-def gla-1 mult-assoc)
  have ad (ad x · y) = ad (ad x · y) · d y + ad (ad x · y) · ad y
    by (metis apd3 a-comm' d1-a' distrib-right' mult-1-right apd-d-def)
  thus ad (ad x · y) ≤ d x + ad y
    by (metis a-subid-aux1' a-join.sup-mono)
next
  have ad y ≤ ad (ad x · y)
    by (simp add: a-antitone' a-subid-aux1')
  thus d x + ad y ≤ ad (ad x · y)
    by (metis apd-d-def a-mul-d' d1-a' gla-1 apd1 join.sup-least mult-assoc)
qed

```

```

lemma d1-sum-var: x + y ≤ (d x + d y) · (x + y)
proof -
  have x + y = d x · x + d y · y
    by simp
  also have ... ≤ (d x + d y) · x + (d x + d y) · y
    using local.distrib-right' local.join.sup-ge1 local.join.sup-ge2 local.join.sup-mono
  by presburger
  finally show ?thesis
    using order-trans subdistl-var by blast
qed

```

```

lemma a4': ad (x + y) = ad x · ad y

```

```

proof (rule antisym)
  show  $ad\ (x + y) \leq ad\ x \cdot ad\ y$ 
    by (metis a-d-closed a-supdist-var add-commute d1-a' local.mult-isol-var)
  hence  $ad\ x \cdot ad\ y = ad\ x \cdot ad\ y + ad\ (x + y)$ 
    using less-eq-def add-commute by simp
  also have  $\dots = ad\ (ad\ (ad\ x \cdot ad\ y) \cdot (x + y))$ 
    by (metis a-closed a-export'')
  finally show  $ad\ x \cdot ad\ y \leq ad\ (x + y)$ 
    using a-antitone' apd-d-def d1-sum-var by auto
qed

```

Antidomain pre-dioids are domain pre-dioids and antidomain near-semirings, but still not antidomain monoids.

```

sublocale dpdz: domain-pre-dioid-one-zero ( $+$ ) ( $\cdot$ )  $1\ 0\ (\leq)\ (<)\ \lambda x.\ ad\ (ad\ x)$ 
  apply (unfold-locales)
  using apd-d-def d1-a' apply auto[1]
  using a2-eq' apd-d-def apply auto[1]
  apply (simp add: a-subid')
  apply (simp add: a4' apd-d-def)
  by (metis a-mul-d' a-very-constrict'' apd-d-def local.mult-onel)

```

```

subclass antidomain-near-semiring
  apply (unfold-locales)
  apply simp
  using local.apd2 local.less-eq-def apply blast
  apply simp
  by (simp add: a4')

```

```

lemma a-supdist:  $ad\ (x + y) \leq ad\ x + ad\ y$ 
  using a-supdist-var local.join.le-supI1 by auto

```

```

lemma a-gla:  $ad\ x \cdot y = 0 \iff ad\ x \leq ad\ y$ 
  using gla-1 a-gla2 by blast

```

```

lemma a-subid-aux2:  $x \cdot ad\ y \leq x$ 
  using a-subid' mult-isol by fastforce

```

```

lemma a42-var:  $d\ x \cdot d\ y \leq ad\ (ad\ x + ad\ y)$ 
  by (simp add: apd-d-def)

```

```

lemma d1-weak [simp]:  $(d\ x + d\ y) \cdot x = x$ 
proof –
  have  $(d\ x + d\ y) \cdot x = (1 + d\ y) \cdot x$ 
    by simp
  thus ?thesis
    by (metis add-commute apd-d-def dpdz.dnso3 local.mult-1-left)
qed

```

```

lemma  $x \cdot ad\ 1 = ad\ 1$ 

```

oops

lemma $ad\ x \cdot (y + z) = ad\ x \cdot y + ad\ x \cdot z$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

lemma $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$

oops

no-notation $apd-d\ (d)$

end

0.15.4 Antidomain Semirings

Antidomain semirings are direct expansions of antidomain pre-dioids, but do not require idempotency of addition. Hence we give a slightly different axiomatisation, following [?].

class *antidomain-semiringl* = *semiring-one-zero*l + *plus-ord* + *antidomain-op* +
assumes *as1* [*simp*]: $ad\ x \cdot x = 0$
and *as2* [*simp*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ y) = ad\ (x \cdot ad\ y)$
and *as3* [*simp*]: $ad\ (ad\ x) + ad\ x = 1$

begin

definition $ads-d :: 'a \Rightarrow 'a\ (d)$ **where**
 $d\ x = ad\ (ad\ x)$

lemma *one-idem'*: $1 + 1 = 1$
by (*metis as1 as2 as3 add-zero mult.right-neutral*)

Every antidomain semiring is a dioid and an antidomain pre-dioid.

subclass *dioid*
by (*standard, metis distrib-left mult.right-neutral one-idem'*)

subclass *antidomain-pre-dioid*
by (*unfold-locales, auto simp: local.less-eq-def*)

lemma *am5-lem* [*simp*]: $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$
proof –
have $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ (x \cdot d\ y) \cdot ad\ (x \cdot ad\ y)$
using *ads-d-def local.a2-eq' local.apd-d-def* **by** *auto*

```

also have ... = ad (x · d y + x · ad y)
  using ans4 by presburger
also have ... = ad (x · (d y + ad y))
  using distrib-left by presburger
finally show ?thesis
  by (simp add: ads-d-def)
qed

lemma am6-lem [simp]: ad (x · y) · x · ad y = ad (x · y) · x
proof -
  fix x y
  have ad (x · y) · x · ad y = ad (x · y) · x · ad y + 0
    by simp
  also have ... = ad (x · y) · x · ad y + ad (x · d y) · x · d y
    using ans1 mult-assoc by presburger
  also have ... = ad (x · y) · x · (ad y + d y)
    using ads-d-def local.a2-eq' local.apd-d-def local.distrib-left by auto
  finally show ad (x · y) · x · ad y = ad (x · y) · x
    using add-commute ads-d-def local.as3 by auto
qed

```

```

lemma a-zero [simp]: ad 0 = 1
  by (simp add: local.a-very-costrict')

```

```

lemma a-one [simp]: ad 1 = 0
  using a-zero local.dpdz.dpd5 by blast

```

```

subclass antidomain-left-monoid
  by (unfold-locales, auto simp: local.a-comm')

```

Every antidomain left semiring is a domain left semiring.

no-notation *domain-semiringl-class*.fd ((|·) ·) [61,81] 82)

definition *fdia* :: 'a ⇒ 'a ⇒ 'a ((|·) ·) [61,81] 82) **where**
 $|x\rangle y = \text{ad } (x \cdot y)$

```

sublocale ds: domain-semiringl (+) (·) 1 0 λx. ad (ad x) (≤) (<)
  rewrites ds.fd x y ≡ fdia x y

```

```

proof -
  show class.domain-semiringl (+) (·) 1 0 (λx. ad (ad x)) (≤) (<)
    by (unfold-locales, auto simp: local.dpdz.dpd4 ans-d-def)
  then interpret ds: domain-semiringl (+) (·) 1 0 λx. ad (ad x) (≤) (<) .
  show ds.fd x y ≡ fdia x y
    by (auto simp: fdia-def ds.fd-def)
qed

```

```

lemma fd-eq-fdia [simp]: domain-semiringl.fd (·) d x y ≡ fdia x y
proof -
  have class.domain-semiringl (+) (·) 1 0 d (≤) (<)

```

```

    by (unfold-locales, auto simp: ads-d-def local.ans-d-def)
  hence domain-semiringl.fd ( $\cdot$ )  $d\ x\ y = d\ ((\cdot)\ x\ y)$ 
    by (rule domain-semiringl.fd-def)
  also have  $\dots = ds.fd\ x\ y$ 
    by (simp add: ds.fd-def ads-d-def)
  finally show domain-semiringl.fd ( $\cdot$ )  $d\ x\ y \equiv |x\rangle\ y$ 
    by auto
qed

end

class antidomain-semiring = antidomain-semiringl + semiring-one-zero

begin

Every antidomain semiring is an antidomain monoid.

subclass antidomain-monoid
  by (standard, metis ans1 mult-1-right annir)

lemma a-zero = 0
  by (simp add: local.a-zero-def)

sublocale ds: domain-semiring (+) ( $\cdot$ ) 1 0  $\lambda x. ad\ (ad\ x)$  ( $\leq$ ) ( $<$ )
  rewrites ds.fd  $x\ y \equiv fdia\ x\ y$ 
  by unfold-locales

end

```

0.15.5 The Boolean Algebra of Domain Elements

```

typedef (overloaded) 'a a2-element = {x :: 'a :: antidomain-semiring. x = d x}
  by (rule-tac x=1 in exI, auto simp: ads-d-def)

setup-lifting type-definition-a2-element

instantiation a2-element :: (antidomain-semiring) boolean-algebra

begin

lift-definition less-eq-a2-element :: 'a a2-element  $\Rightarrow$  'a a2-element  $\Rightarrow$  bool is ( $\leq$ )
.

lift-definition less-a2-element :: 'a a2-element  $\Rightarrow$  'a a2-element  $\Rightarrow$  bool is ( $<$ ) .

lift-definition bot-a2-element :: 'a a2-element is 0
  by (simp add: ads-d-def)

lift-definition top-a2-element :: 'a a2-element is 1
  by (simp add: ads-d-def)

```

lift-definition $\text{inf-a2-element} :: 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element}$
is (\cdot)

by $(\text{metis } (\text{no-types}, \text{lifting}) \text{ ads-d-def dpdz.dom-mult-closed})$

lift-definition $\text{sup-a2-element} :: 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element}$
is $(+)$

by $(\text{metis } \text{ads-d-def ds.dsr5})$

lift-definition $\text{minus-a2-element} :: 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element}$
is $\lambda x y. x \cdot \text{ad } y$

by $(\text{metis } (\text{no-types}, \text{lifting}) \text{ ads-d-def dpdz.domain-export''})$

lift-definition $\text{uminus-a2-element} :: 'a \text{ a2-element} \Rightarrow 'a \text{ a2-element}$ **is** antidomain-op
by $(\text{simp add: ads-d-def})$

instance

apply $(\text{standard}; \text{transfer})$

apply $(\text{simp add: less-le-not-le})$

apply simp

apply $\text{auto}[1]$

apply simp

apply $(\text{metis } \text{a-subid-aux2 ads-d-def})$

apply $(\text{metis } \text{a-subid-aux1' ads-d-def})$

apply $(\text{metis } (\text{no-types}, \text{lifting}) \text{ ads-d-def dpdz.dom-glb})$

apply simp

apply simp

apply simp

apply simp

apply $(\text{metis } \text{a-subid' ads-d-def})$

apply $(\text{metis } (\text{no-types}, \text{lifting}) \text{ ads-d-def dpdz.dom-distrib})$

apply $(\text{metis } \text{ads-d-def ans1})$

apply $(\text{metis } \text{ads-d-def ans3})$

by simp

end

0.15.6 Further Properties

context $\text{antidomain-semiringl}$

begin

lemma $\text{a-2-var: } \text{ad } x \cdot d \ y = 0 \longleftrightarrow \text{ad } x \leq \text{ad } y$

using $\text{local.a-gla local.ads-d-def local.dpdz.dom-weakly-local}$ **by** auto

The following two lemmas give the Galois connection of Heyting algebras.

lemma $\text{da-shunt1: } x \leq d \ y + z \implies x \cdot \text{ad } y \leq z$

proof –

```

assume  $x \leq d\ y + z$ 
hence  $x \cdot ad\ y \leq (d\ y + z) \cdot ad\ y$ 
  using mult-isor by blast
also have  $\dots = d\ y \cdot ad\ y + z \cdot ad\ y$ 
  by simp
also have  $\dots \leq z$ 
  by (simp add: a-subid-aux2 ads-d-def)
finally show  $x \cdot ad\ y \leq z$ 
  by simp
qed

```

```

lemma da-shunt2:  $x \leq ad\ y + z \implies x \cdot d\ y \leq z$ 
  using da-shunt1 local.a-add-idem local.ads-d-def am-add-op-def by auto

```

```

lemma d-a-galois1:  $d\ x \cdot ad\ y \leq d\ z \iff d\ x \leq d\ z + d\ y$ 
  by (metis add-assoc local.a-gla local.ads-d-def local.am2 local.ans4 local.ans-d-def
    local.dnsz.dnso4)

```

```

lemma d-a-galois2:  $d\ x \cdot d\ y \leq d\ z \iff d\ x \leq d\ z + ad\ y$ 
proof –
  have  $\bigwedge a\ aa. ad\ ((a::'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$ 
    using local.a2-eq' local.apd-d-def by force
  then show ?thesis
    by (metis d-a-galois1 local.a-export' local.ads-d-def local.ans-d-def)
qed

```

```

lemma d-cancellation-1:  $d\ x \leq d\ y + d\ x \cdot ad\ y$ 
proof –
  have  $a: d\ (d\ x \cdot ad\ y) = ad\ y \cdot d\ x$ 
    using local.a-closure' local.ads-d-def local.am2 local.ans-d-def by auto
  hence  $d\ x \leq d\ (d\ x \cdot ad\ y) + d\ y$ 
    using d-a-galois1 local.a-comm-var local.ads-d-def by fastforce
  thus ?thesis
    using a-add-commute local.ads-d-def local.am2 by auto
qed

```

```

lemma d-cancellation-2:  $(d\ z + d\ y) \cdot ad\ y \leq d\ z$ 
  by (simp add: da-shunt1)

```

```

lemma a-de-morgan:  $ad\ (ad\ x \cdot ad\ y) = d\ (x + y)$ 
  by (simp add: local.ads-d-def)

```

```

lemma a-de-morgan-var-3:  $ad\ (d\ x + d\ y) = ad\ x \cdot ad\ y$ 
  using local.a-add-idem local.ads-d-def am-add-op-def by auto

```

```

lemma a-de-morgan-var-4:  $ad\ (d\ x \cdot d\ y) = ad\ x + ad\ y$ 
  using local.a-add-idem local.ads-d-def am-add-op-def by auto

```

```

lemma a-4:  $ad\ x \leq ad\ (x \cdot y)$ 

```


using *local.a-add-idem local.a-antitone' local.dpdz.domain-1'' am-add-op-def* **by** *fastforce*

lemma *a-6*: $ad (d x \cdot y) = ad x + ad y$
using *a-de-morgan-var-4 local.ads-d-def* **by** *auto*

lemma *a-7*: $d x \cdot ad (d y + d z) = d x \cdot ad y \cdot ad z$
using *a-de-morgan-var-3 local.mult.semigroup-axioms semigroup.assoc* **by** *fastforce*

lemma *a-d-add-closure [simp]*: $d (ad x + ad y) = ad x + ad y$
using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *d-6 [simp]*: $d x + ad x \cdot d y = d x + d y$
proof –
have $ad (ad x \cdot (x + ad y)) = d (x + y)$
by (*simp add: distrib-left ads-d-def*)
thus *?thesis*
by (*simp add: local.ads-d-def local.ans-d-def*)
qed

lemma *d-7 [simp]*: $ad x + d x \cdot ad y = ad x + ad y$
by (*metis a-d-add-closure local.ads-d-def local.ans4 local.s4*)

lemma *a-mult-add*: $ad x \cdot (y + x) = ad x \cdot y$
by (*simp add: distrib-left*)

lemma *kat-2*: $y \cdot ad z \leq ad x \cdot y \implies d x \cdot y \cdot ad z = 0$
proof –
assume *a*: $y \cdot ad z \leq ad x \cdot y$
hence $d x \cdot y \cdot ad z \leq d x \cdot ad x \cdot y$
using *local.mult-isol mult-assoc* **by** *presburger*
thus *?thesis*
using *local.join.le-bot ads-d-def* **by** *auto*
qed

lemma *kat-3*: $d x \cdot y \cdot ad z = 0 \implies d x \cdot y = d x \cdot y \cdot d z$
using *local.a-zero-def local.ads-d-def local.am-d-def local.kat-3'* **by** *auto*

lemma *kat-4*: $d x \cdot y = d x \cdot y \cdot d z \implies d x \cdot y \leq y \cdot d z$
using *a-subid-aux1 mult-assoc ads-d-def* **by** *auto*

lemma *kat-2-equiv*: $y \cdot ad z \leq ad x \cdot y \iff d x \cdot y \cdot ad z = 0$
proof
assume $y \cdot ad z \leq ad x \cdot y$
thus $d x \cdot y \cdot ad z = 0$
by (*simp add: kat-2*)
next
assume *1*: $d x \cdot y \cdot ad z = 0$
have $y \cdot ad z = (d x + ad x) \cdot y \cdot ad z$

```

    by (simp add: local.ads-d-def)
  also have ... = d x · y · ad z + ad x · y · ad z
    using local.distrib-right by presburger
  also have ... = ad x · y · ad z
    using 1 by auto
  also have ... ≤ ad x · y
    by (simp add: local.a-subid-aux2)
  finally show y · ad z ≤ ad x · y .
qed

```

```

lemma kat-4-equiv: d x · y = d x · y · d z ⟷ d x · y ≤ y · d z
  using local.ads-d-def local.dpdz.d-preserves-equation by auto

```

```

lemma kat-3-equiv-opp: ad z · y · d x = 0 ⟷ y · d x = d z · y · d x
proof -
  have ad z · (y · d x) = 0 ⟶ (ad z · y · d x = 0) = (y · d x = d z · y · d x)
    by (metis (no-types, hide-lams) add-commute local.add-zero local.ads-d-def
      local.as3 local.distrib-right' local.mult-1-left mult-assoc)
  thus ?thesis
    by (metis a-4 local.a-add-idem local.a-gla2 local.ads-d-def mult-assoc am-add-op-def)
qed

```

```

lemma kat-4-equiv-opp: y · d x = d z · y · d x ⟷ y · d x ≤ d z · y
  using kat-2-equiv kat-3-equiv-opp local.ads-d-def by auto

```

0.15.7 Forward Box and Diamond Operators

```

lemma fdemodalisation22: |x⟩ y ≤ d z ⟷ ad z · x · d y = 0

```

```

proof -
  have |x⟩ y ≤ d z ⟷ d (x · y) ≤ d z
    by (simp add: fdia-def ads-d-def)
  also have ... ⟷ ad z · d (x · y) = 0
    by (metis add-commute local.a-gla local.ads-d-def local.ans4)
  also have ... ⟷ ad z · x · y = 0
    using dpdz.dom-weakly-local mult-assoc ads-d-def by auto
  finally show ?thesis
    using dpdz.dom-weakly-local ads-d-def by auto
qed

```

```

lemma dia-diff-var: |x⟩ y ≤ |x⟩ (d y · ad z) + |x⟩ z

```

```

proof -
  have 1: |x⟩ (d y · d z) ≤ |x⟩ (1 · d z)
    using dpdz.dom-glb-eq ds.fd-subdist fdia-def ads-d-def by force
  have |x⟩ y = |x⟩ (d y · (ad z + d z))
    by (metis as3 add-comm ds.fdia-d-simp mult-1-right ads-d-def)
  also have ... = |x⟩ (d y · ad z) + |x⟩ (d y · d z)
    by (simp add: local.distrib-left local.ds.fdia-add1)
  also have ... ≤ |x⟩ (d y · ad z) + |x⟩ (1 · d z)
    using 1 local.join.sup.mono by blast

```

finally show *?thesis*
by (*simp add: fdia-def ads-d-def*)
qed

lemma *dia-diff*: $|x\rangle y \cdot \text{ad } (|x\rangle z) \leq |x\rangle (d y \cdot \text{ad } z)$
using *fdia-def dia-diff-var d-a-galois2 ads-d-def* **by** *metis*

lemma *fdia-export-2*: $\text{ad } y \cdot |x\rangle z = |\text{ad } y \cdot x\rangle z$
using *local.am-d-def local.d-a-export local.fdia-def mult-assoc* **by** *auto*

lemma *fdia-split*: $|x\rangle y = d z \cdot |x\rangle y + \text{ad } z \cdot |x\rangle y$
by (*metis mult-onel ans3 distrib-right ads-d-def*)

definition *fbox* :: $'a \Rightarrow 'a \Rightarrow 'a \ ((|-) \text{ } -) \ [61,81] \ 82)$ **where**
 $|x] y = \text{ad } (x \cdot \text{ad } y)$

The next lemmas establish the De Morgan duality between boxes and diamonds.

lemma *fdia-fbox-de-morgan-2*: $\text{ad } (|x\rangle y) = |x] \text{ad } y$
using *fbox-def local.a-closure local.a-loc local.am-d-def local.fdia-def* **by** *auto*

lemma *fbox-simp*: $|x] y = |x] d y$
using *fbox-def local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *fbox-dom* [*simp*]: $|x] 0 = \text{ad } x$
by (*simp add: fbox-def*)

lemma *fbox-add1*: $|x] (d y \cdot d z) = |x] y \cdot |x] z$
using *a-de-morgan-var-4 fbox-def local.distrib-left* **by** *auto*

lemma *fbox-add2*: $|x + y] z = |x] z \cdot |y] z$
by (*simp add: fbox-def*)

lemma *fbox-mult*: $|x \cdot y] z = |x] |y] z$
using *fbox-def local.a2-eq' local.apd-d-def mult-assoc* **by** *auto*

lemma *fbox-zero* [*simp*]: $|0] x = 1$
by (*simp add: fbox-def*)

lemma *fbox-one* [*simp*]: $|1] x = d x$
by (*simp add: fbox-def ads-d-def*)

lemma *fbox-iso*: $d x \leq d y \implies |z] x \leq |z] y$

proof –

assume $d x \leq d y$

hence $\text{ad } y \leq \text{ad } x$

using *local.a-add-idem local.a-antitone' local.ads-d-def am-add-op-def* **by** *fastforce*

hence $z \cdot \text{ad } y \leq z \cdot \text{ad } x$

by (*simp add: mult-isol*)

thus $|z| \ x \leq |z| \ y$
by (*simp add: fbox-def a-antitone'*)
qed

lemma *fbox-antitone-var*: $x \leq y \implies |y| \ z \leq |x| \ z$
by (*simp add: fbox-def a-antitone mult-isor*)

lemma *fbox-subdist-1*: $|x| \ (d \ y \cdot d \ z) \leq |x| \ y$
using *a-de-morgan-var-4 fbox-def local.a-supdist-var local.distrib-left* **by** *force*

lemma *fbox-subdist-2*: $|x| \ y \leq |x| \ (d \ y + d \ z)$
by (*simp add: fbox-iso ads-d-def*)

lemma *fbox-subdist*: $|x| \ d \ y + |x| \ d \ z \leq |x| \ (d \ y + d \ z)$
by (*simp add: fbox-iso sup-least ads-d-def*)

lemma *fbox-diff-var*: $|x| \ (d \ y + ad \ z) \cdot |x| \ z \leq |x| \ y$
proof –
have $ad \ (ad \ y) \cdot ad \ (ad \ z) = ad \ (ad \ z + ad \ y)$
using *local.dpdz.dsg4* **by** *auto*
then have $d \ (d \ (d \ y + ad \ z) \cdot d \ z) \leq d \ y$
by (*simp add: local.a-subid-aux1' local.ads-d-def*)
then show *?thesis*
by (*metis fbox-add1 fbox-iso*)
qed

lemma *fbox-diff*: $|x| \ (d \ y + ad \ z) \leq |x| \ y + ad \ (|x| \ z)$
proof –
have $f1: \bigwedge a. ad \ (ad \ (ad \ (a::'a))) = ad \ a$
using *local.a-closure' local.ans-d-def* **by** *force*
have $f2: \bigwedge a \ aa. ad \ (ad \ (a::'a)) + ad \ aa = ad \ (ad \ a \cdot aa)$
using *local.ans-d-def* **by** *auto*
have $f3: \bigwedge a \ aa. ad \ ((a::'a) + aa) = ad \ (aa + a)$
by (*simp add: local.am2*)
then have $f4: \bigwedge a \ aa. ad \ (ad \ (ad \ (a::'a) \cdot aa)) = ad \ (ad \ aa + a)$
using $f2 \ f1$ **by** (*metis (no-types) local.ans4*)
have $f5: \bigwedge a \ aa \ ab. ad \ ((a::'a) \cdot (aa + ab)) = ad \ (a \cdot (ab + aa))$
using $f3 \ local.distrib-left$ **by** *presburger*
have $f6: \bigwedge a \ aa. ad \ (ad \ (ad \ (a::'a) + aa)) = ad \ (ad \ aa \cdot a)$
using $f3 \ f1$ **by** *fastforce*
have $ad \ (x \cdot ad \ (y + ad \ z)) \leq ad \ (ad \ (x \cdot ad \ z) \cdot (x \cdot ad \ y))$
using $f5 \ f2 \ f1$ **by** (*metis (no-types) a-mult-add fbox-def fbox-subdist-1 local.a-gla2 local.ads-d-def local.ans4 local.distrib-left local.gla-1 mult-assoc*)
then show *?thesis*
using $f6 \ f4 \ f3 \ f1$ **by** (*simp add: fbox-def local.ads-d-def*)
qed

end

context *antidomain-semiring*

begin

lemma *kat-1*: $d\ x \cdot y \leq y \cdot d\ z \implies y \cdot ad\ z \leq ad\ x \cdot y$

proof –

assume *a*: $d\ x \cdot y \leq y \cdot d\ z$

have $y \cdot ad\ z = d\ x \cdot y \cdot ad\ z + ad\ x \cdot y \cdot ad\ z$

by (*metis local.ads-d-def local.as3 local.distrib-right local.mult-1-left*)

also have $\dots \leq y \cdot (d\ z \cdot ad\ z) + ad\ x \cdot y \cdot ad\ z$

by (*metis a add-iso mult-isor mult-assoc*)

also have $\dots = ad\ x \cdot y \cdot ad\ z$

by (*simp add: ads-d-def*)

finally show $y \cdot ad\ z \leq ad\ x \cdot y$

using *local.a-subid-aux2 local.dual-order.trans* **by** *blast*

qed

lemma *kat-1-equiv*: $d\ x \cdot y \leq y \cdot d\ z \longleftrightarrow y \cdot ad\ z \leq ad\ x \cdot y$

using *kat-1 kat-2 kat-3 kat-4* **by** *blast*

lemma *kat-3-equiv'*: $d\ x \cdot y \cdot ad\ z = 0 \longleftrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$

by (*simp add: kat-1-equiv local.kat-2-equiv local.kat-4-equiv*)

lemma *kat-1-equiv-opp*: $y \cdot d\ x \leq d\ z \cdot y \longleftrightarrow ad\ z \cdot y \leq y \cdot ad\ x$

by (*metis kat-1-equiv local.a-closure' local.ads-d-def local.ans-d-def*)

lemma *kat-2-equiv-opp*: $ad\ z \cdot y \leq y \cdot ad\ x \longleftrightarrow ad\ z \cdot y \cdot d\ x = 0$

by (*simp add: kat-1-equiv-opp local.kat-3-equiv-opp local.kat-4-equiv-opp*)

lemma *fbox-one-1* [*simp*]: $|x| \cdot 1 = 1$

by (*simp add: fbox-def*)

lemma *fbox-demodalisation3*: $d\ y \leq |x| \cdot d\ z \longleftrightarrow d\ y \cdot x \leq x \cdot d\ z$

by (*simp add: fbox-def a-gla kat-2-equiv-opp mult-assoc ads-d-def*)

end

0.15.8 Antidomain Kleene Algebras

class *antidomain-left-kleene-algebra* = *antidomain-semiringl* + *left-kleene-algebra-zero*

begin

sublocale *dka*: *domain-left-kleene-algebra* (+) (\cdot) 1 0 *d* (\leq) ($<$) *star*

rewrites *domain-semiringl.fd* (\cdot) $d\ x\ y \equiv |x\rangle\ y$

by (*unfold-locales, auto simp add: local.ads-d-def ans-d-def*)

lemma *a-star* [*simp*]: $ad\ (x^\star) = 0$

using *dka.dom-star local.a-very-costrict'' local.ads-d-def* **by** *force*

```

lemma fbox-star-unfold [simp]:  $|1| \ z \cdot |x| \ |x^*| \ z = |x^*| \ z$ 
proof –
  have  $ad \ (ad \ z + x \cdot (x^* \cdot ad \ z)) = ad \ (x^* \cdot ad \ z)$ 
    using local.conway.dagger-unfoldl-distr mult-assoc by auto
  then show ?thesis
    using local.a-closure' local.ans-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2
by fastforce
qed

```

```

lemma fbox-star-unfold-var [simp]:  $d \ z \cdot |x| \ |x^*| \ z = |x^*| \ z$ 
using fbox-star-unfold by auto

```

```

lemma fbox-star-unfoldr [simp]:  $|1| \ z \cdot |x^*| \ |x| \ z = |x^*| \ z$ 
by (metis fbox-star-unfold fbox-mult star-slide-var)

```

```

lemma fbox-star-unfoldr-var [simp]:  $d \ z \cdot |x^*| \ |x| \ z = |x^*| \ z$ 
using fbox-star-unfoldr by auto

```

```

lemma fbox-star-induct-var:  $d \ y \leq |x| \ y \implies d \ y \leq |x^*| \ y$ 
proof –
  assume a1:  $d \ y \leq |x| \ y$ 
  have  $\bigwedge a. ad \ (ad \ (ad \ (a::'a))) = ad \ a$ 
    using local.a-closure' local.ans-d-def by auto
  then have  $ad \ (ad \ (x^* \cdot ad \ y)) \leq ad \ y$ 
    using a1 by (metis dka.fdia-star-induct local.a-export' local.ads-d-def local.ans4
local.ans-d-def local.eq-refl local.fbox-def local.fdia-def local.meet-ord-def)
  then have  $ad \ (ad \ y + ad \ (x^* \cdot ad \ y)) = zero-class.zero$ 
    by (metis (no-types) add-commute local.a-2-var local.ads-d-def local.ans4)
  then show ?thesis
    using local.a-2-var local.ads-d-def local.fbox-def by auto
qed

```

```

lemma fbox-star-induct:  $d \ y \leq d \ z \cdot |x| \ y \implies d \ y \leq |x^*| \ z$ 
proof –
  assume a1:  $d \ y \leq d \ z \cdot |x| \ y$ 
  hence a:  $d \ y \leq d \ z$  and  $d \ y \leq |x| \ y$ 
    apply (metis local.a-subid-aux2 local.dual-order.trans local.fbox-def)
    using a1 dka.dom-subid-aux2 local.dual-order.trans by blast
  hence  $d \ y \leq |x^*| \ y$ 
    using fbox-star-induct-var by blast
  thus ?thesis
    using a local.fbox-iso local.order.trans by blast
qed

```

```

lemma fbox-star-induct-eq:  $d \ z \cdot |x| \ y = d \ y \implies d \ y \leq |x^*| \ z$ 
by (simp add: fbox-star-induct)

```

```

lemma fbox-export-1:  $ad \ y + |x| \ y = |d \ y \cdot x| \ y$ 

```

```

    by (simp add: local.a-6 local.fbox-def mult-assoc)

lemma fbox-export-2:  $d\ y + \lfloor x \rfloor\ y = \lfloor ad\ y \cdot x \rfloor\ y$ 
  by (simp add: local.ads-d-def local.ans-d-def local.fbox-def mult-assoc)

end

class antidomain-kleene-algebra = antidomain-semiring + kleene-algebra

begin

subclass antidomain-left-kleene-algebra ..

lemma  $d\ p \leq \lfloor (d\ t \cdot x)^* \cdot ad\ t \rfloor\ (d\ q \cdot ad\ t) \implies d\ p \leq \lfloor d\ t \cdot x \rfloor\ d\ q$ 

oops

end

end

```

0.16 Range and Antirange Semirings

```

theory Range-Semiring
imports Antidomain-Semiring
begin

```

0.16.1 Range Semirings

We set up the duality between domain and antidomain semirings on the one hand and range and antirange semirings on the other hand.

```

class range-op =
  fixes range-op :: 'a  $\Rightarrow$  'a (r)

class range-semiring = semiring-one-zero + plus-ord + range-op +
  assumes rsr1 [simp]:  $x + (x \cdot r\ x) = x \cdot r\ x$ 
  and rsr2 [simp]:  $r\ (r\ x \cdot y) = r\ (x \cdot y)$ 
  and rsr3 [simp]:  $r\ x + 1 = 1$ 
  and rsr4 [simp]:  $r\ 0 = 0$ 
  and rsr5 [simp]:  $r\ (x + y) = r\ x + r\ y$ 

begin

definition bd :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a ( $\langle - \mid - \rangle$  [61,81] 82) where
   $\langle x \mid y = r\ (y \cdot x)$ 

no-notation range-op (r)

```

end

sublocale *range-semiring* \subseteq *rdual: domain-semiring* (+) $\lambda x y. y \cdot x$ 1 0 *range-op* (\leq) ($<$)
 rewrites *rdual.fd* $x y = \langle x | y$
proof –
 show *class.domain-semiring* (+) $(\lambda x y. y \cdot x)$ 1 0 *range-op* (\leq) ($<$)
 by (*standard*, *auto simp: mult-assoc distrib-left*)
 then **interpret** *rdual: domain-semiring* (+) $\lambda x y. y \cdot x$ 1 0 *range-op* (\leq) ($<$) .
 show *rdual.fd* $x y = \langle x | y$
 unfolding *rdual.fd-def* *bd-def* **by** *auto*
qed

sublocale *domain-semiring* \subseteq *ddual: range-semiring* (+) $\lambda x y. y \cdot x$ 1 0 *domain-op* (\leq) ($<$)
 rewrites *ddual.bd* $x y = \text{domain-semiringl-class.fd } x y$
proof –
 show *class.range-semiring* (+) $(\lambda x y. y \cdot x)$ 1 0 *domain-op* (\leq) ($<$)
 by (*standard*, *auto simp: mult-assoc distrib-left*)
 then **interpret** *ddual: range-semiring* (+) $\lambda x y. y \cdot x$ 1 0 *domain-op* (\leq) ($<$) .
 show *ddual.bd* $x y = \text{domain-semiringl-class.fd } x y$
 unfolding *ddual.bd-def* *fd-def* **by** *auto*
qed

0.16.2 Antirange Semirings

class *antirange-op* =
 fixes *antirange-op* :: 'a \Rightarrow 'a (*ar* - [999] 1000)
class *antirange-semiring* = *semiring-one-zero* + *plus-ord* + *antirange-op* +
 assumes *ars1* [*simp*]: $x \cdot \text{ar } x = 0$
 and *ars2* [*simp*]: $\text{ar } (x \cdot y) + \text{ar } (\text{ar } ar \ x \cdot y) = \text{ar } (\text{ar } ar \ x \cdot y)$
 and *ars3* [*simp*]: $\text{ar } ar \ x + \text{ar } x = 1$

begin

no-notation *bd* ($\langle - |$ - [61,81] 82)

definition *ars-r* :: 'a \Rightarrow 'a (*r*) **where**
 $r \ x = \text{ar } (\text{ar } x)$

definition *bdia* :: 'a \Rightarrow 'a \Rightarrow 'a ($\langle - |$ - [61,81] 82) **where**
 $\langle x | y = \text{ar } ar \ (y \cdot x)$

definition *bbox* :: 'a \Rightarrow 'a \Rightarrow 'a ($[- |$ - [61,81] 82) **where**
 $[x | y = \text{ar } (\text{ar } y \cdot x)$

end


```

sublocale antirange-semiring  $\subseteq$  ardual: antidomain-semiring antirange-op (+)  $\lambda x$ 
 $y. y \cdot x$  1 0 ( $\leq$ ) ( $<$ )
  rewrites ardual.ads-d  $x = r\ x$ 
  and ardual.fdia  $x\ y = \langle x \mid y$ 
  and ardual.fbox  $x\ y = [x \mid y$ 
proof –
  show class.antidomain-semiring antirange-op (+) ( $\lambda x\ y. y \cdot x$ ) 1 0 ( $\leq$ ) ( $<$ )
    by (standard, auto simp: mult-assoc distrib-left)
  then interpret ardual: antidomain-semiring antirange-op (+)  $\lambda x\ y. y \cdot x$  1 0
( $\leq$ ) ( $<$ ) .
  show ardual.ads-d  $x = r\ x$ 
    by (simp add: ardual.ads-d-def local.ars-r-def)
  show ardual.fdia  $x\ y = \langle x \mid y$ 
    unfolding ardual.fdia-def bdia-def by auto
  show ardual.fbox  $x\ y = [x \mid y$ 
    unfolding ardual.fbox-def bbox-def by auto
qed

context antirange-semiring

begin

sublocale rs: range-semiring (+) ( $\cdot$ ) 1 0  $\lambda x. ar\ ar\ x$  ( $\leq$ ) ( $<$ )
  by unfold-locales

end

sublocale antidomain-semiring  $\subseteq$  addual: antirange-semiring (+)  $\lambda x\ y. y \cdot x$  1 0
antidomain-op ( $\leq$ ) ( $<$ )
  rewrites addual.ars-r  $x = d\ x$ 
  and addual.bdia  $x\ y = |x\rangle\ y$ 
  and addual.bbox  $x\ y = |x]\ y$ 
proof –
  show class.antirange-semiring (+) ( $\lambda x\ y. y \cdot x$ ) 1 0 antidomain-op ( $\leq$ ) ( $<$ )
    by (standard, auto simp: mult-assoc distrib-left)
  then interpret addual: antirange-semiring (+)  $\lambda x\ y. y \cdot x$  1 0 antidomain-op
( $\leq$ ) ( $<$ ) .
  show addual.ars-r  $x = d\ x$ 
    by (simp add: addual.ars-r-def local.ads-d-def)
  show addual.bdia  $x\ y = |x\rangle\ y$ 
    unfolding addual.bdia-def fdia-def by auto
  show addual.bbox  $x\ y = |x]\ y$ 
    unfolding addual.bbox-def fbox-def by auto
qed

```

0.16.3 Antirange Kleene Algebras

```

class antirange-kleene-algebra = antirange-semiring + kleene-algebra

```

sublocale *antirange-kleene-algebra* \subseteq *dual: antidomain-kleene-algebra antirange-op*
 (+) $\lambda x y. y \cdot x \ 1 \ 0 \ (\leq) \ (<) \ star$
 by (*standard, auto simp: local.star-inductr' local.star-inductl*)

sublocale *antidomain-kleene-algebra* \subseteq *dual: antirange-kleene-algebra* (+) $\lambda x y. y$
 $\cdot x \ 1 \ 0 \ (\leq) \ (<) \ star \ antidomain-op$
 by (*standard, simp-all add: star-inductr star-inductl*)

Hence all range theorems have been derived by duality in a generic way.

end

0.17 Modal Kleene Algebras

This section studies laws that relate antidomain and antirange semirings and Kleene algebra, notably Galois connections and conjugations as those studied in [?, ?].

theory *Modal-Kleene-Algebra*
imports *Range-Semiring*
begin

class *modal-semiring* = *antidomain-semiring* + *antirange-semiring* +
assumes *domrange* [*simp*]: $d \ (r \ x) = r \ x$
and *rangedom* [*simp*]: $r \ (d \ x) = d \ x$

begin

These axioms force that the domain algebra and the range algebra coincide.

lemma *domrangefix*: $d \ x = x \longleftrightarrow r \ x = x$
 by (*metis domrange rangedom*)

lemma *box-diamond-galois-1*:
assumes $d \ p = p$ **and** $d \ q = q$
shows $\langle x | \ p \leq q \longleftrightarrow p \leq |x \rangle \ q$
proof –
 have $\langle x | \ p \leq q \longleftrightarrow p \cdot x \leq x \cdot q$
 by (*metis assms domrangefix local.ardual.ds.fdemodalisation2 local.ars-r-def*)
 thus ?thesis
 by (*metis assms fbox-demodalisation3*)
qed

lemma *box-diamond-galois-2*:
assumes $d \ p = p$ **and** $d \ q = q$
shows $|x \rangle \ p \leq q \longleftrightarrow p \leq [x] \ q$
proof –
 have $|x \rangle \ p \leq q \longleftrightarrow x \cdot p \leq q \cdot x$
 by (*metis assms local.ads-d-def local.ds.fdemodalisation2*)

thus *?thesis*
by (*metis* *assms* *domrange**fix* *local.ardual.fbox-demodalisation3*)
qed

lemma *diamond-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $|x\rangle\ p \leq ad\ q \longleftrightarrow \langle x|\ q \leq ad\ p$
proof –
have $|x\rangle\ p \leq ad\ q \longleftrightarrow x \cdot p \leq ad\ q \cdot x$
by (*metis* *assms* *local.ads-d-def* *local.ds.fdemodalisation2*)
also have $\dots \longleftrightarrow q \cdot x \leq x \cdot ad\ p$
by (*metis* *assms* *local.ads-d-def* *local.kat-1-equiv-opp*)
finally show *?thesis*
by (*metis* *assms* *box-diamond-galois-1* *local.ads-d-def* *local.fbox-demodalisation3*)
qed

lemma *diamond-conjugation-aux*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $\langle x|\ p \leq ad\ q \longleftrightarrow q \cdot \langle x|\ p = 0$
apply *standard*
apply (*metis* *assms*(2) *local.a-antitone'* *local.a-gla* *local.ads-d-def*)
proof –
assume $a1: q \cdot \langle x|\ p = \text{zero-class.zero}$
have $ad\ (ad\ q) = q$
using *assms*(2) *local.ads-d-def* **by** *fastforce*
then show $\langle x|\ p \leq ad\ q$
using *a1* **by** (*metis* (*no-types*) *domrange**fix* *local.a-gla* *local.ads-d-def* *local.antisym*
local.ardual.a-gla2 *local.ardual.gla-1* *local.ars-r-def* *local.bdia-def* *local.eq-refl*)
qed

lemma *diamond-conjugation*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $p \cdot |x\rangle\ q = 0 \longleftrightarrow q \cdot \langle x|\ p = 0$
proof –
have $p \cdot |x\rangle\ q = 0 \longleftrightarrow |x\rangle\ q \leq ad\ p$
by (*metis* *assms*(1) *local.a-gla* *local.ads-d-def* *local.am2* *local.fdia-def*)
also have $\dots \longleftrightarrow \langle x|\ p \leq ad\ q$
by (*simp* *add: assms* *diamond-conjugation-var-1*)
finally show *?thesis*
by (*simp* *add: assms* *diamond-conjugation-aux*)
qed

lemma *box-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $ad\ p \leq [x]\ q \longleftrightarrow ad\ q \leq [x]\ p$
by (*metis* *assms* *box-diamond-galois-1* *box-diamond-galois-2* *diamond-conjugation-var-1*
local.ads-d-def)

lemma *box-diamond-cancellation-1*: $d\ p = p \implies p \leq [x]\ \langle x|\ p$

```

using box-diamond-galois-1 domrangefix local.ars-r-def local.bdia-def by fastforce

lemma box-diamond-cancellation-2:  $d\ p = p \implies p \leq [x \mid x] \ p$ 
by (metis box-diamond-galois-2 local.ads-d-def local.dpdz.domain-invol local.eq-refl
local.fdia-def)

lemma box-diamond-cancellation-3:  $d\ p = p \implies [x] \ [x] \ p \leq p$ 
using box-diamond-galois-2 domrangefix local.ardual.fdia-fbox-de-morgan-2 lo-
cal.ars-r-def local.bbox-def local.bdia-def by auto

lemma box-diamond-cancellation-4:  $d\ p = p \implies \langle x \mid [x] \ p \leq p$ 
using box-diamond-galois-1 local.ads-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2
by auto

end

class modal-kleene-algebra = modal-semiring + kleene-algebra
begin

subclass antidomain-kleene-algebra ..

subclass antirange-kleene-algebra ..

end

end

```

0.18 Models of Modal Kleene Algebras

```

theory Modal-Kleene-Algebra-Models
imports Kleene-Algebra-Models Modal-Kleene-Algebra

begin

```

This section develops the relation model. We also briefly develop the trace model for antidomain Kleene algebras, but not for antirange or full modal Kleene algebras. The reason is that traces are implemented as lists; we therefore expect tedious inductive proofs in the presence of range. The language model is not particularly interesting.

```

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

```

```

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

```

```

definition trace-a :: ('p, 'a) trace set  $\Rightarrow$  ('p, 'a) trace set where
  trace-a  $X = \{(p, []) \mid p. \neg (\exists x. x \in X \wedge p = first\ x)\}$ 

```

interpretation *trace-antidomain-kleene-algebra*: *antidomain-kleene-algebra trace-a*

(\cup) *t-prod t-one t-zero* (\subseteq) (\subset) *t-star*

proof

show $\bigwedge x. t\text{-prod } (trace\text{-}a\ x) x = t\text{-zero}$

by (*auto simp: trace-a-def t-prod-def t-fusion-def t-zero-def*)

show $\bigwedge x\ y. trace\text{-}a\ (t\text{-prod } x\ y) \cup trace\text{-}a\ (t\text{-prod } x\ (trace\text{-}a\ (trace\text{-}a\ y))) = trace\text{-}a\ (t\text{-prod } x\ (trace\text{-}a\ (trace\text{-}a\ y)))$

by (*auto simp: trace-a-def t-prod-def t-fusion-def*)

show $\bigwedge x. trace\text{-}a\ (trace\text{-}a\ x) \cup trace\text{-}a\ x = t\text{-one}$

by (*auto simp: trace-a-def t-one-def*)

qed

The trace model should be extended to cover modal Kleene algebras in the future.

definition *rel-ar* :: '*a rel* \Rightarrow '*a rel* **where**

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

interpretation *rel-antirange-kleene-algebra*: *antirange-semiring* (\cup) (*O*) *Id* $\{\}$ *rel-ar*

(\subseteq) (\subset)

apply *unfold-locales*

apply (*simp-all add: rel-ar-def*)

by *auto*

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

(\subset) *rtrancl rel-ad rel-ar*

apply *standard*

apply (*metis* (*no-types, lifting*) *rel-antidomain-kleene-algebra.a-d-closed rel-antidomain-kleene-algebra.a-one*

rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem

rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.apd-d-def rel-antidomain-kleene-algebra.dka.dns1

rel-antidomain-kleene-algebra.dpdz.dom-one rel-antirange-kleene-algebra.ardual.a-comm'

rel-antirange-kleene-algebra.ardual.a-d-closed rel-antirange-kleene-algebra.ardual.a-mul-d'

rel-antirange-kleene-algebra.ardual.a-mult-idem rel-antirange-kleene-algebra.ardual.a-zero

rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardual.am6-lem

rel-antirange-kleene-algebra.ardual.apd-d-def rel-antirange-kleene-algebra.ardual.s4)

by (*metis* *rel-antidomain-kleene-algebra.a-zero rel-antidomain-kleene-algebra.addual.ars1*

rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem

rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.ds.ddual.mult-oner

rel-antidomain-kleene-algebra.s4 rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardual.am

rel-antirange-kleene-algebra.ardual.apd1 rel-antirange-kleene-algebra.ardual.dpdz.dns1')

end

0.19 Components Based on Kleene Algebra with Domain

theory *VC-KAD*

imports *Modal-Kleene-Algebra-Models Algebraic-VCs.P2S2R*

begin

0.19.1 Verification Component for Backward Reasoning

This component supports the verification of simple while programs in a partial correctness setting.

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

no-notation *Archimedean-Field.floor* ($\lfloor \cdot \rfloor$)

notation *p2r* ($\lceil \cdot \rceil$)

notation *r2p* ($\lfloor \cdot \rfloor$)

context *antidomain-kleene-algebra*

begin

Additional Facts for KAD

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*)

Syntax for Conditionals and Loops

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

definition *while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - od* $[64,64]$ 63) **where**
 $while\ p\ do\ x\ od = (d\ p \cdot x)^* \cdot ad\ p$

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

Basic Weakest (Liberal) Precondition Calculus

In the setting of Kleene algebra with domain, the wlp operator is the forward modal box operator of antidomain Kleene algebra.

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-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-export3:  $|x \cdot ad\ p| \ q = |x| \ (d\ p + d\ q)$ 
  using a-de-morgan-var-3 ds.ddual.mult-assoc fbox-def by auto

```

```

lemma fbox-seq [simp]:  $|x \cdot y| \ q = |x| \ |y| \ q$ 
  by (simp add: fbox-mult)

```

```

lemma fbox-seq-var:  $p' \leq |y| \ q \implies p \leq |x| \ p' \implies p \leq |x \cdot y| \ q$ 
proof -
  assume h1:  $p \leq |x| \ p'$  and h2:  $p' \leq |y| \ q$ 
  hence  $|x| \ p' \leq |x| \ |y| \ q$ 
    by (simp add: dka.dom-iso fbox-iso)
  thus ?thesis
    by (metis h1 dual-order.trans fbox-seq)
qed

```

```

lemma fbox-cond-var [simp]:  $|if\ p\ then\ x\ else\ y\ fi| \ q = (ad\ p + |x| \ q) \cdot (d\ p + |y| \ q)$ 
  using cond-def a-closure' ads-d-def ans-d-def fbox-add2 fbox-export1 by auto

```

```

lemma fbox-cond-aux1 [simp]:  $d\ p \cdot |if\ p\ then\ x\ else\ y\ fi| \ q = d\ p \cdot |x| \ q$ 
proof -
  have  $d\ p \cdot |if\ p\ then\ x\ else\ y\ fi| \ q = d\ p \cdot |x| \ q \cdot (d\ p + d\ (|y| \ q))$ 
    using a-d-add-closure addual.ars-r-def ds.ddual.mult-assoc fbox-def fbox-cond-var
  by auto
  thus ?thesis
    by (metis addual.ars-r-def am2 dka.dns5 ds.ddual.mult-assoc fbox-def)
qed

```

```

lemma fbox-cond-aux2 [simp]:  $ad\ p \cdot |if\ p\ then\ x\ else\ y\ fi| \ q = ad\ p \cdot |y| \ q$ 
  by (metis cond-def a-closure' add-commute addual.ars-r-def ans-d-def fbox-cond-aux1)

```

```

lemma fbox-cond [simp]:  $|if\ p\ then\ x\ else\ y\ fi| \ q = (d\ p \cdot |x| \ q) + (ad\ p \cdot |y| \ q)$ 
proof -
  have  $|if\ p\ then\ x\ else\ y\ fi| \ q = (d\ p + ad\ p) \cdot |if\ p\ then\ x\ else\ y\ fi| \ q$ 
    by (simp add: addual.ars-r-def)
  thus ?thesis
    by (metis distrib-right' fbox-cond-aux1 fbox-cond-aux2)
qed

```

```

lemma fbox-cond-var2 [simp]:  $|if\ p\ then\ x\ else\ y\ fi| \ q = if\ p\ then\ |x| \ q\ else\ |y| \ q\ fi$ 
  using cond-def fbox-cond by auto

```

lemma *fbox-while-unfold*: $|while\ t\ do\ x\ od|\ p = (d\ t + d\ p) \cdot (ad\ t + |x|\ |while\ t\ do\ x\ od|\ p)$
by (*metis fbox-export1 fbox-export3 dka.dom-add-closed fbox-star-unfold-var while-def*)

lemma *fbox-while-var1*: $d\ t \cdot |while\ t\ do\ x\ od|\ p = d\ t \cdot |x|\ |while\ t\ do\ x\ od|\ p$
by (*metis fbox-while-unfold a-mult-add ads-d-def dka.dns5 ds.ddual.mult-assoc join.sup-commute*)

lemma *fbox-while-var2*: $ad\ t \cdot |while\ t\ do\ x\ od|\ p \leq d\ p$

proof –

have $ad\ t \cdot |while\ t\ do\ x\ od|\ p = ad\ t \cdot (d\ t + d\ p) \cdot (ad\ t + |x|\ |while\ t\ do\ x\ od|\ p)$

by (*metis fbox-while-unfold ds.ddual.mult-assoc*)

also have $\dots = ad\ t \cdot d\ p \cdot (ad\ t + |x|\ |while\ t\ do\ x\ od|\ p)$

by (*metis a-de-morgan-var-3 addual.ars-r-def dka.dom-add-closed s4*)

also have $\dots \leq d\ p \cdot (ad\ t + |x|\ |while\ t\ do\ x\ od|\ p)$

using *a-subid-aux1' mult-isor* **by** *blast*

finally show *?thesis*

by (*metis a-de-morgan-var-3 a-mult-idem addual.ars-r-def ans4 dka.dsr5 dpdz.domain-1'' dual-order.trans fbox-def*)

qed

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

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 while-def fbox-export2* **by** *presburger*

qed

lemma *fbox-while-var*: $d\ p = |d\ t \cdot x|\ p \implies d\ p \leq |while\ t\ do\ x\ od|\ (d\ p \cdot ad\ t)$

by (*metis eq-refl fbox-export1 fbox-shunt fbox-while*)

lemma *fbox-whilei*: $d\ p \leq d\ i \implies d\ i \cdot ad\ t \leq d\ q \implies d\ i \cdot d\ t \leq |x|\ i \implies d\ p \leq |while\ t\ inv\ i\ do\ x\ od|\ q$

proof –

assume *a1*: $d\ p \leq d\ i$ **and** *a2*: $d\ i \cdot ad\ t \leq d\ q$ **and** $d\ i \cdot d\ t \leq |x|\ i$

hence $d\ i \leq |while\ t\ inv\ i\ do\ x\ od|\ (d\ i \cdot ad\ t)$

by (*simp add: fbox-while whilei-def*)

also have $\dots \leq |while\ t\ inv\ i\ do\ x\ od|\ q$

using *a2 dka.dom-iso fbox-iso* **by** *fastforce*

finally show *?thesis*

using *a1 dual-order.trans* **by** *blast*

qed

The next rule is used for dealing with while loops after a series of sequential

steps.

lemma *fbox-whilei-break*: $d\ p \leq |y|\ i \implies d\ i \cdot ad\ t \leq d\ q \implies d\ i \cdot d\ t \leq |x|\ i$
 $\implies d\ p \leq |y| \cdot (while\ t\ inv\ i\ do\ x\ od)]\ q$
apply (*rule fbox-seq-var*, *rule fbox-whilei*, *simp-all*, *blast*)
using *fbox-simp* **by** *auto*

Finally we derive a frame rule.

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

lemma *fbox-frame-var*: $d\ p \leq |x|\ p \implies d\ q \leq |x|\ t \implies d\ p \cdot d\ q \leq |x|\ (d\ p \cdot d\ t)$
using *fbox-frame fbox-demodalisation3 fbox-simp* **by** *auto*

end

Store and Assignment

type-synonym *'a store* = *string* \Rightarrow *'a*

notation *rel-antidomain-kleene-algebra.fbox* (*wp*)
and *rel-antidomain-kleene-algebra.fdia* (*relfdia*)

definition *gets* :: *string* \Rightarrow (*'a store* \Rightarrow *'a*) \Rightarrow *'a store* *rel* (*-* ::= *-* [70, 65] 61)
where
 $v ::= e = \{(s, s\ (v ::= e\ s)) \mid s. \text{True}\}$

lemma *assign-prop*: $\lceil \lambda s. P\ (s\ (v ::= e\ s)) \rceil ; (v ::= e) = (v ::= e) ; \lceil P \rceil$
by (*auto simp add: p2r-def gets-def*)

lemma *wp-assign* [*simp*]: $wp\ (v ::= e)\ \lceil Q \rceil = \lceil \lambda s. Q\ (s\ (v ::= e\ s)) \rceil$
by (*auto simp: rel-antidomain-kleene-algebra.fbox-def gets-def rel-ad-def p2r-def*)

lemma *wp-assign-var* [*simp*]: $\lfloor wp\ (v ::= e)\ \lceil Q \rceil \rfloor = (\lambda s. Q\ (s\ (v ::= e\ s)))$
by (*simp, auto simp: r2p-def p2r-def*)

lemma *wp-assign-det*: $wp\ (v ::= e)\ \lceil Q \rceil = relfdia\ (v ::= e)\ \lceil Q \rceil$
by (*auto simp add: rel-antidomain-kleene-algebra.fbox-def rel-antidomain-kleene-algebra.fdia-def gets-def p2r-def rel-ad-def, fast*)

Simplifications

notation *rel-antidomain-kleene-algebra.ads-d* (*rdom*)

abbreviation *spec-sugar* :: *'a pred* \Rightarrow *'a rel* \Rightarrow *'a pred* \Rightarrow *bool* (*PRE* - - *POST* -
[64, 64, 64] 63) **where**
 $PRE\ P\ X\ POST\ Q \equiv rdom\ \lceil P \rceil \subseteq wp\ X\ \lceil Q \rceil$

abbreviation *cond-sugar* :: 'a pred \Rightarrow 'a rel \Rightarrow 'a rel \Rightarrow 'a rel (IF - THEN - ELSE - FI [64,64,64] 63) **where**

IF P THEN X ELSE Y FI \equiv rel-antidomain-kleene-algebra.cond $\lceil P \rceil$ X Y

abbreviation *whilei-sugar* :: 'a pred \Rightarrow 'a pred \Rightarrow 'a rel \Rightarrow 'a rel (WHILE - INV - DO - OD [64,64,64] 63) **where**

WHILE P INV I DO X OD \equiv rel-antidomain-kleene-algebra.whilei $\lceil P \rceil$ $\lceil I \rceil$ X

lemma *d-p2r [simp]*: rdom $\lceil P \rceil$ = $\lceil P \rceil$

by (simp add: p2r-def rel-antidomain-kleene-algebra.ads-d-def rel-ad-def)

lemma *p2r-conj-hom-var-symm [simp]*: $\lceil P \rceil$; $\lceil Q \rceil$ = $\lceil P \sqcap Q \rceil$

by (simp add: p2r-conj-hom-var)

lemma *p2r-neg-hom [simp]*: rel-ad $\lceil P \rceil$ = $\lceil - P \rceil$

by (simp add: rel-ad-def p2r-def)

lemma *wp-trafo*: $\lfloor wp X \lceil Q \rceil \rfloor$ = $(\lambda s. \forall s'. (s, s') \in X \longrightarrow Q s')$

by (auto simp: rel-antidomain-kleene-algebra.fbox-def rel-ad-def p2r-def r2p-def)

lemma *wp-trafo-var*: $\lfloor wp X \lceil Q \rceil \rfloor s$ = $(\forall s'. (s, s') \in X \longrightarrow Q s')$

by (simp add: wp-trafo)

lemma *wp-simp*: rdom $\lfloor wp X Q \rfloor$ = wp X Q

by (metis d-p2r rel-antidomain-kleene-algebra.a-subid' rel-antidomain-kleene-algebra.addual.bbox-def rpr)

lemma *wp-simp-var [simp]*: wp $\lceil P \rceil$ $\lceil Q \rceil$ = $\lceil - P \sqcup Q \rceil$

by (simp add: rel-antidomain-kleene-algebra.fbox-def)

lemma *impl-prop [simp]*: $\lceil P \rceil \subseteq \lceil Q \rceil \longleftrightarrow (\forall s. P s \longrightarrow Q s)$

by (auto simp: p2r-def)

lemma *p2r-eq-prop [simp]*: $\lceil P \rceil = \lceil Q \rceil \longleftrightarrow (\forall s. P s \longleftrightarrow Q s)$

by (auto simp: p2r-def)

lemma *impl-prop-var [simp]*: rdom $\lceil P \rceil \subseteq$ rdom $\lceil Q \rceil \longleftrightarrow (\forall s. P s \longrightarrow Q s)$

by simp

lemma *p2r-eq-prop-var [simp]*: rdom $\lceil P \rceil =$ rdom $\lceil Q \rceil \longleftrightarrow (\forall s. P s \longleftrightarrow Q s)$

by simp

lemma *wp-whilei*: $(\forall s. P s \longrightarrow I s) \implies (\forall s. (I \sqcap -T) s \longrightarrow Q s) \implies (\forall s. (I \sqcap T) s \longrightarrow \lfloor wp X \lceil I \rceil \rfloor s)$

$\implies (\forall s. P s \longrightarrow \lfloor wp (WHILE T INV I DO X OD) \lceil Q \rceil \rfloor s)$

apply (simp only: impl-prop-var[symmetric] wp-simp)

by (rule rel-antidomain-kleene-algebra.fbox-whilei, simp-all, simp-all add: p2r-def)

end

0.20 VC_diffKAD

```

theory VC-diffKAD-auxiliarities
imports
  Main
  ../afpModified/VC-KAD
  Ordinary-Differential-Equations.ODE-Analysis

```

```

begin

```

0.20.1 Stack Theories Preliminaries: VC_KAD and ODEs

To make our notation less code-like and more mathematical we declare:

```

no-notation Archimedean-Field.ceiling ( $\lceil \cdot \rceil$ )
  and Archimedean-Field.floor ( $\lfloor \cdot \rfloor$ )
  and Set.image (  $'$  )
  and Range-Semiring.antirange-semiring-class.ars-r (  $r$  )

notation p2r ( $\lceil \cdot \rceil$ )
  and r2p ( $\lfloor \cdot \rfloor$ )
  and Set.image ( $-\lceil \cdot \rceil$ )
  and Product-Type.prod.fst ( $\pi_1$ )
  and Product-Type.prod.snd ( $\pi_2$ )
  and List.zip (infixl  $\otimes$  63)
  and rel-ad ( $\Delta^c_1$ )

```

This and more notation is explained by the following lemmata.

```

lemma shows  $\lceil P \rceil = \{(s, s) \mid s. P\ s\}$ 
  and  $\lfloor R \rfloor = (\lambda x. x \in r2s\ R)$ 
  and  $r2s\ R = \{x \mid x. \exists y. (x, y) \in R\}$ 
  and  $\pi_1\ (x, y) = x \wedge \pi_2\ (x, y) = y$ 
  and  $\Delta^c_1\ R = \{(x, x) \mid x. \nexists y. (x, y) \in R\}$ 
  and  $wp\ R\ Q = \Delta^c_1\ (R ; \Delta^c_1\ Q)$ 
  and  $[x1, x2, x3, x4] \otimes [y1, y2] = [(x1, y1), (x2, y2)]$ 
  and  $\{a..b\} = \{x. a \leq x \wedge x \leq b\}$ 
  and  $\{a <..< b\} = \{x. a < x \wedge x < b\}$ 
  and  $(x\ solves\ ode\ f)\ \{0..t\}\ R = ((x\ has\ vderiv\ on\ (\lambda t. f\ t\ (x\ t)))\ \{0..t\} \wedge x \in \{0..t\} \rightarrow R)$ 
  and  $f \in A \rightarrow B = (f \in \{f. \forall x. x \in A \longrightarrow (f\ x) \in B\})$ 
  and  $(x\ has\ vderiv\ on\ x')\ \{0..t\} =$ 
     $(\forall r \in \{0..t\}. (x\ has\ vector\ derivative\ x'\ r)\ (at\ r\ within\ \{0..t\}))$ 
  and  $(x\ has\ vector\ derivative\ x'\ r)\ (at\ r\ within\ \{0..t\}) =$ 
     $(x\ has\ derivative\ (\lambda x. x *_{R} x'\ r))\ (at\ r\ within\ \{0..t\})$ 
apply(simp-all add: p2r-def r2p-def rel-ad-def rel-antidomain-kleene-algebra.fbox-def

  solves-ode-def has-vderiv-on-def)
apply(blast, fastforce, fastforce)
using has-vector-derivative-def by auto

```

Observe also, the following consequences and facts:

proposition $\pi_1(\downarrow R) = r2s\ R$
by (*simp add: fst-eq-Domain*)

proposition $\Delta^c_1\ R = Id - \{(s, s) \mid s. s \in (\pi_1(\downarrow R))\}$
by(*simp add: image-def rel-ad-def, fastforce*)

proposition $P \subseteq Q \implies wp\ R\ P \subseteq wp\ R\ Q$
by(*simp add: rel-antidomain-kleene-algebra.dka.dom-iso rel-antidomain-kleene-algebra.fbox-iso*)

proposition *boxProgrPred-IsProp*: $wp\ R\ \lceil P \rceil \subseteq Id$
by(*simp add: rel-antidomain-kleene-algebra.a-subid' rel-antidomain-kleene-algebra.addual.bbox-def*)

proposition *rdom-p2r-contents*: $(a, b) \in rdom\ \lceil P \rceil = ((a = b) \wedge P\ a)$
proof–
have $(a, b) \in rdom\ \lceil P \rceil = ((a = b) \wedge (a, a) \in rdom\ \lceil P \rceil)$ **using** *p2r-subid* **by**
fastforce
also have $\dots = ((a = b) \wedge (a, a) \in \lceil P \rceil)$ **by** *simp*
also have $\dots = ((a = b) \wedge P\ a)$ **by** (*simp add: p2r-def*)
ultimately show *?thesis* **by** *simp*
qed

~~//Should not add these complement rule's to simp//~~
proposition *rel-ad-rule1*: $(x, x) \notin \Delta^c_1\ \lceil P \rceil \implies P\ x$
by(*auto simp: rel-ad-def p2r-subid p2r-def*)

proposition *rel-ad-rule2*: $(x, x) \in \Delta^c_1\ \lceil P \rceil \implies \neg P\ x$
by(*metis ComplD VC-KAD.p2r-neg-hom rel-ad-rule1 empty-iff mem-Collect-eq p2s-neg-hom*
rel-antidomain-kleene-algebra.a-one rel-antidomain-kleene-algebra.am1 relcomp.relcompI)

proposition *rel-ad-rule3*: $R \subseteq Id \implies (x, x) \notin R \implies (x, x) \in \Delta^c_1\ R$
by(*metis IdI Un-iff d-p2r rel-antidomain-kleene-algebra.addual.ars3*
rel-antidomain-kleene-algebra.addual.ars-r-def rpr)

proposition *rel-ad-rule4*: $(x, x) \in R \implies (x, x) \notin \Delta^c_1\ R$
by(*metis empty-iff rel-antidomain-kleene-algebra.addual.ars1 relcomp.relcompI*)

proposition *boxProgrPred-chrctrztn*: $(x, x) \in wp\ R\ \lceil P \rceil = (\forall\ y. (x, y) \in R \longrightarrow P\ y)$
by(*metis boxProgrPred-IsProp rel-ad-rule1 rel-ad-rule2 rel-ad-rule3*
rel-ad-rule4 d-p2r wp-simp wp-trafo)

lemma (*in antidomain-kleene-algebra*) *fbox-starI*:
assumes $d\ p \leq d\ i$ **and** $d\ i \leq |x|\ i$ **and** $d\ i \leq d\ q$
shows $d\ p \leq |x^*|\ q$
proof–
from $\langle d\ i \leq |x|\ i \rangle$ **have** $d\ i \leq |x|\ i$ (*d i*)
using *local.fbox-simp* **by** *auto*

hence $|1| \ p \leq |x^*| \ i$ **using** $\langle d \ p \leq d \ i \rangle$ **by** $(metis \ (no-types) \ local.dual-order.trans \ local.fbox-one \ local.fbox-simp \ local.fbox-star-induct-var)$
thus $?thesis$ **using** $\langle d \ i \leq d \ q \rangle$ **by** $(metis \ (full-types) \ local.fbox-mult \ local.fbox-one \ local.fbox-seq-var \ local.fbox-simp)$
qed

proposition *cons-eq-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-zip-left-rightD*:

$(x, y) \in set \ (xList \otimes yList) \implies x \in set \ xList \wedge y \in set \ yList$

apply $(rule \ conjI)$

apply $(rule-tac \ y=y \ \text{and} \ ys=yList \ \text{in} \ set-zip-leftD, \ simp)$

apply $(rule-tac \ x=x \ \text{and} \ xs=xList \ \text{in} \ set-zip-rightD, \ simp)$

done

declare *zip-map-fst-snd* $[simp]$

0.20.2 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 $varDiffs\text{-}to\text{-}zero :: real\ store \Rightarrow real\ store\ (sol)\ \text{where}$
 $sol\ a \equiv (override\text{-}on\ a\ (\lambda\ x.\ 0)\ varDiffs)$

proposition $varDiffs\text{-}to\text{-}zero\text{-}vdiff[simp]: (sol\ s)\ (\partial\ x) = 0$
apply($simp\ add: override\text{-}on\text{-}def\ varDiffs\text{-}def$)
by *auto*

proposition $varDiffs\text{-}to\text{-}zero\text{-}beginning[simp]: take\ 2\ x \neq ''d['' \Longrightarrow (sol\ s)\ x = s$
 x
apply($simp\ add: varDiffs\text{-}def\ override\text{-}on\text{-}def\ vdiff\text{-}def$)
by *fastforce*

— Next, for each entry of the input-list, we update the state using said entry.

definition $vderiv\text{-}of\ f\ S = (SOME\ f'.\ (f\ has\text{-}vderiv\text{-}on\ f')\ S)$

primrec $state\text{-}list\text{-}upd :: ((real \Rightarrow real\ store \Rightarrow real) \times string \times (real\ store \Rightarrow real))\ list \Rightarrow$
 $real \Rightarrow real\ store \Rightarrow real\ store\ \text{where}$
 $state\text{-}list\text{-}upd\ []\ t\ s = s$
 $state\text{-}list\text{-}upd\ (uxf\ \# tail)\ t\ s = (state\text{-}list\text{-}upd\ tail\ t\ s)$
 $(\quad (\pi_1\ (\pi_2\ uxf)) := (\pi_1\ uxf)\ t\ s,$
 $\quad \partial\ (\pi_1\ (\pi_2\ uxf)) := (if\ t = 0\ then\ (\pi_2\ (\pi_2\ uxf))\ s$
 $else\ vderiv\text{-}of\ (\lambda\ r.\ (\pi_1\ uxf)\ r\ s)\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t))$

abbreviation $state\text{-}list\text{-}cross\text{-}upd :: real\ store \Rightarrow (string \times (real\ store \Rightarrow real))\ list$
 \Rightarrow
 $(real \Rightarrow real\ store \Rightarrow real)\ list \Rightarrow real \Rightarrow (char\ list \Rightarrow real)\ (-[\leftarrow] - [64, 64, 64]$
 $63)\ \text{where}$
 $s[xfList \leftarrow uInput]\ t \equiv state\text{-}list\text{-}upd\ (uInput \otimes xfList)\ t\ s$

proposition $state\text{-}list\text{-}cross\text{-}upd\text{-}empty[simp]: (s[\leftarrow list]\ t) = s$
by(*induction list, simp-all*)

lemma *inductive-state-list-cross-upd-its-vars:*

assumes $distHyp: distinct\ (map\ \pi_1\ ((y, g) \# xftail))$
and $varHyp: \forall\ xf \in set((y, g) \# xftail). \pi_1\ xf \notin varDiffs$
and $indHyp: (u, x, f) \in set\ (utail \otimes xftail) \Longrightarrow (s[xftail \leftarrow utail]\ t)\ x = u\ t\ s$
and $disjHyp: (u, x, f) = (v, y, g) \vee (u, x, f) \in set\ (utail \otimes xftail)$
shows $(s[(y, g) \# xftail \leftarrow v \# utail]\ t)\ x = u\ t\ s$
using *disjHyp proof*
assume $(u, x, f) = (v, y, g)$
hence $(s[(y, g) \# xftail \leftarrow v \# utail]\ t)\ x = ((s[xftail \leftarrow utail]\ t)(x := u\ t\ s,$
 $\partial\ x := if\ t = 0\ then\ f\ s\ else\ vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0 <..< (2 *_{\mathbb{R}} t)\}\ t))\ x\ \text{by}$

simp
 also have $\dots = u \ t \ s$ by (*simp add: vdiff-def*)
 ultimately show *?thesis* by *simp*
 next
 assume $yTailHyp: (u, x, f) \in \text{set } (uTail \otimes xTail)$
 from *this* and *indHyp* have $\exists: (s[xTail \leftarrow uTail] \ t) \ x = u \ t \ s$ by *fastforce*
 from *yTailHyp* and *distHyp* have $\exists: y \neq x$ using *set-zip-left-rightD* by *force*
 from *yTailHyp* and *varHyp* have $\exists: x \neq \partial \ y$
 using *set-zip-left-rightD vdiff-invarDiffs* by *fastforce*
 from 1 and 2 have $(s[(y, g) \# xTail \leftarrow v \# uTail] \ t) \ x = (s[xTail \leftarrow uTail] \ t) \ x$
 by *simp*
 thus *?thesis* using 3 by *simp*
 qed

theorem *state-list-cross-upd-its-vars*:
 assumes *distinctHyp*: *distinct* (*map* π_1 *xfList*)
 and *lengthHyp*: *length* *xfList* = *length* *uInput*
 and *varsHyp*: $\forall \ xf \in \text{set } xfList. \ \pi_1 \ xf \notin \text{varDiffs}$
 and *its-var*: $(u, x, f) \in \text{set } (uInput \otimes xfList)$
 shows $(s[xfList \leftarrow uInput] \ t) \ x = u \ t \ s$
 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*: $x \in X \implies (\text{override-on } f \ g \ X)(x := z) = (\text{override-on } f \ (g(x := z))) \ X$
 by (*rule ext*, *simp add: override-on-def*)

lemma *inductive-state-list-cross-upd-its-dvars*:
 assumes $\exists \ g. (s[xfTail \leftarrow uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
 and $\forall \ xf \in \text{set } (xf \ \# \ xfTail). \ \pi_1 \ xf \notin \text{varDiffs}$
 and $\forall \ uxf \in \text{set } (u \ \# \ uTail \otimes xf \ \# \ xfTail). \ \pi_1 \ uxf \ 0 \ s = s \ (\pi_1 \ (\pi_2 \ uxf))$
 shows $\exists \ g. (s[xf \ \# \ xfTail \leftarrow u \ \# \ uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
 proof—
 let *?gLHS* = $(s[(xf \ \# \ xfTail) \leftarrow (u \ \# \ uTail)] \ 0)$
 have *observ*: $\partial \ (\pi_1 \ xf) \in \text{varDiffs}$ by (*auto simp: varDiffs-def*)
 from *assms*(1) obtain *g* where $(s[xfTail \leftarrow uTail] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$
 by *force*
 then have *?gLHS* = $(\text{override-on } s \ g \ \text{varDiffs})(\pi_1 \ xf := u \ 0 \ s, \ \partial \ (\pi_1 \ xf) := \pi_2 \ xf \ s)$ by *simp*
 also have $\dots = (\text{override-on } s \ g \ \text{varDiffs})(\partial \ (\pi_1 \ xf) := \pi_2 \ xf \ s)$
 using *override-on-def* *varDiffs-def* *assms* by *auto*
 also have $\dots = (\text{override-on } s \ (g(\partial \ (\pi_1 \ xf) := \pi_2 \ xf \ s)) \ \text{varDiffs})$
 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 $\text{varsHyp}:\forall \text{xf} \in \text{set } \text{xfList}. \pi_1 \text{xf} \notin \text{varDiffs}$
and $\text{solHyp1}:\forall \text{uxf} \in \text{set } (\text{uInput} \otimes \text{xfList}). (\pi_1 \text{uxf}) \ 0 \ s = s \ (\pi_1 \ (\pi_2 \ \text{uxf}))$
shows $\exists g. (s[\text{xfList} \leftarrow \text{uInput}] \ 0) = (\text{override-on } s \ g \ \text{varDiffs})$
using assms **proof**($\text{induct } \text{xfList } \text{uInput}$ rule: list-induct2')
case 1
 have $(s[\] \leftarrow \] \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
 unfolding override-on-def **by** simp
 thus $?case$ **by** metis
next
 case (2 $\text{xf } \text{xfTail}$)
 have $(s[\text{xf} \ \# \ \text{xfTail} \leftarrow \] \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
 unfolding override-on-def **by** simp
 thus $?case$ **by** metis
next
 case (3 $u \ \text{utail}$)
 have $(s[\] \leftarrow \ \text{utail}] \ 0) = \text{override-on } s \ s \ \text{varDiffs}$
 unfolding override-on-def **by** simp
 thus $?case$ **by** force
next
 case (4 $\text{xf } \text{xfTail } u \ \text{uTail}$)
 then have $\exists g. (s[\text{xfTail} \leftarrow \ \text{uTail}] \ 0) = \text{override-on } s \ g \ \text{varDiffs}$ **by** simp
 thus $?case$ **using** $\text{inductive-state-list-cross-upd-its-dvars } 4.\text{prems}$ **by** blast
qed

lemma $\text{vderiv-unique-within-open-interval}$:
assumes $(f \ \text{has-vderiv-on } f') \ \{0 < .. < t\}$ **and** $t > 0$
 and $(f \ \text{has-vderiv-on } f'') \ \{0 < .. < t\}$ **and** $\text{tauHyp}:\tau \in \{0 < .. < t\}$
shows $f' \ \tau = f'' \ \tau$
using assms **apply**($\text{simp add: has-vderiv-on-def has-vector-derivative-def}$)
using $\text{frechet-derivative-unique-within-open-interval}$ **by** ($\text{metis box-real}(1) \ \text{scaleR-one } \text{tauHyp}$)

lemma $\text{has-vderiv-on-cong-open-interval}$:
assumes $g\text{Hyp}:\forall \tau > 0. f \ \tau = g \ \tau$ **and** $t\text{Hyp}: t > 0$
and $f\text{Hyp}:(f \ \text{has-vderiv-on } f') \ \{0 < .. < t\}$
shows $(g \ \text{has-vderiv-on } f') \ \{0 < .. < t\}$
proof–
from $g\text{Hyp}$ **have** $\bigwedge \tau. \tau \in \{0 < .. < t\} \implies f \ \tau = g \ \tau$ **using** $t\text{Hyp}$ **by** force
hence $\text{eqDs}:(f \ \text{has-vderiv-on } f') \ \{0 < .. < t\} = (g \ \text{has-vderiv-on } f') \ \{0 < .. < t\}$
apply($\text{rule-tac has-vderiv-on-cong}$) **by** auto
thus $(g \ \text{has-vderiv-on } f') \ \{0 < .. < t\}$ **using** $\text{eqDs } f\text{Hyp}$ **by** simp
qed

lemma $\text{closed-vderiv-on-cong-to-open-vderiv}$:
assumes $g\text{Hyp}:\forall \tau > 0. f \ \tau = g \ \tau$
and $f\text{Hyp}:\forall t \geq 0. (f \ \text{has-vderiv-on } f') \ \{0 .. t\}$
and $t\text{Hyp}: t > 0$ **and** $c\text{Hyp}: c > 1$
shows $\text{vderiv-of } g \ \{0 < .. < (c *_{\mathbb{R}} t)\} \ t = f' \ t$
proof–

have $ctHyp:c \cdot t > 0$ **using** $tHyp$ **and** $cHyp$ **by** *auto*
from $fHyp$ **have** $(f \text{ has-vderiv-on } f') \{0 < .. < c \cdot t\}$ **using** *has-vderiv-on-subset*
by *(metis greaterThanLessThan-subseteq-atLeastAtMost-iff less-eq-real-def)*
then have $derivHyp:(g \text{ has-vderiv-on } f') \{0 < .. < c \cdot t\}$
using $gHyp$ $ctHyp$ **and** *has-vderiv-on-cong-open-interval* **by** *blast*
hence $f'Hyp:\forall f''. (g \text{ has-vderiv-on } f'') \{0 < .. < c \cdot t\} \longrightarrow (\forall \tau \in \{0 < .. < c \cdot t\}.$
 $f' \tau = f'' \tau)$
using *vderiv-unique-within-open-interval* $ctHyp$ **by** *blast*
also have $(g \text{ has-vderiv-on } (vderiv\text{-of } g \{0 < .. < (c *_R t)\})) \{0 < .. < c \cdot t\}$
by*(simp add: vderiv-of-def, metis derivHyp someI-ex)*
ultimately show $vderiv\text{-of } g \{0 < .. < c *_R t\} t = f' t$ **using** $tHyp$ $cHyp$ **by** *force*
qed

lemma *vderiv-of-to-sol-its-vars*:
assumes $distinctHyp:distinct (\text{map } \pi_1 \text{ } xfList)$
and $lengthHyp:length \text{ } xfList = length \text{ } uInput$
and $varsHyp:\forall xf \in \text{set } xfList. \pi_1 \text{ } xf \notin \text{varDiffs}$
and $solHyp2:\forall t \geq 0. ((\lambda \tau. (sol \text{ } s[xfList \leftarrow uInput] \text{ } \tau) \text{ } x)$
 $\text{has-vderiv-on } (\lambda \tau. f (sol \text{ } s[xfList \leftarrow uInput] \text{ } \tau))) \{0..t\}$
and $tHyp:t > 0$ **and** $uxfHyp:(u, x, f) \in \text{set } (uInput \otimes xfList)$
shows $vderiv\text{-of } (\lambda \tau. u \text{ } \tau (sol \text{ } s)) \{0 < .. < (2 *_R t)\} t = f (sol \text{ } s[xfList \leftarrow uInput]$
 $t)$
apply $(rule\text{-tac } f = (\lambda \tau. (sol \text{ } s[xfList \leftarrow uInput] \text{ } \tau) \text{ } x))$ **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 $eqFuncs:\forall s. \forall g. \forall xf \in \text{set } ((x, f) \# xfs). \pi_2 \text{ } xf (override\text{-on } s \text{ } g \text{ } \text{varDiffs})$
 $= \pi_2 \text{ } xf \text{ } s$
and $eqLengths:length ((x, f) \# xfs) = length (u \# us)$
and $distinct:distinct (\text{map } \pi_1 ((x, f) \# xfs))$
and $vars:\forall xf \in \text{set } ((x, f) \# xfs). \pi_1 \text{ } xf \notin \text{varDiffs}$
and $solHyp1:\forall uxf \in \text{set } ((u \# us) \otimes ((x, f) \# xfs)). \pi_1 \text{ } uxf \text{ } 0 (sol \text{ } s) = sol \text{ } s (\pi_1$
 $(\pi_2 \text{ } uxf))$
and $disjHyp:(y, g) = (x, f) \vee (y, g) \in \text{set } xfs$
and $indHyp:(y, g) \in \text{set } xfs \implies (sol \text{ } s[xfs \leftarrow us] \text{ } 0) (\partial \text{ } y) = g (sol \text{ } s[xfs \leftarrow us] \text{ } 0)$
shows $(sol \text{ } s[(x, f) \# xfs \leftarrow u \# us] \text{ } 0) (\partial \text{ } y) = g (sol \text{ } s[(x, f) \# xfs \leftarrow u \# us] \text{ } 0)$
proof–
from *assms* **obtain** $h1$ **where** $h1Def:(sol \text{ } s[((x, f) \# xfs) \leftarrow (u \# us)] \text{ } 0) =$
 $(override\text{-on } (sol \text{ } s) \text{ } h1 \text{ } \text{varDiffs})$ **using** *state-list-cross-upd-its-dvars* **by** *blast*
from $disjHyp$ **show** $(sol \text{ } s[(x, f) \# xfs \leftarrow u \# us] \text{ } 0) (\partial \text{ } y) = g (sol \text{ } s[(x, f) \#$
 $xfs \leftarrow u \# us] \text{ } 0)$
proof
assume $eqHeads:(y, g) = (x, f)$
then have $g (sol \text{ } s[(x, f) \# xfs \leftarrow u \# us] \text{ } 0) = f (sol \text{ } s)$ **using** $h1Def$ $eqFuncs$
by *simp*
also have $... = (sol \text{ } s[(x, f) \# xfs \leftarrow u \# us] \text{ } 0) (\partial \text{ } y)$ **using** $eqHeads$ **by** *auto*
ultimately show *?thesis* **by** *linarith*
next

assume $\text{tailHyp}:(y, g) \in \text{set } xfs$
 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 $h2\text{Def}:(\text{sol } s[xfs \leftarrow us] \ 0) = \text{override-on } (\text{sol } s) \ h2 \ \text{varDiffs}$
 using *state-list-cross-upd-its-dvars eqLengths distinct vars* and solHyp1 by force
 have $(\text{sol } s[(x, f) \# xfs \leftarrow u \# us] \ 0) (\partial y) = g (\text{sol } s[xfs \leftarrow us] \ 0)$
 using tailHyp *indHyp* $\langle x \neq \partial y \rangle$ and $\langle \partial x \neq \partial y \rangle$ by *simp*
 also have $\dots = g (\text{override-on } (\text{sol } s) \ h2 \ \text{varDiffs})$ using $h2\text{Def}$ by *simp*
 also have $\dots = g (\text{sol } s)$ using *eqFuncs* and tailHyp by force
 also have $\dots = g (\text{sol } s[(x, f) \# xfs \leftarrow u \# us] \ 0)$
 using *eqFuncs h1Def tailHyp* and *eq-snd-iff* by *fastforce*
 ultimately show *?thesis* by *simp*
 qed
 qed

lemma *to-sol-zero-its-dvars*:
 assumes $\text{funcsHyp}:\forall s. \forall g. \forall xf \in \text{set } xfList. \pi_2 \ xf \ (\text{override-on } s \ g \ \text{varDiffs})$
 $= \pi_2 \ xf \ s$
 and $\text{distinctHyp}:\text{distinct } (\text{map } \pi_1 \ xfList)$
 and $\text{lengthHyp}:\text{length } xfList = \text{length } uInput$
 and $\text{varsHyp}:\forall xf \in \text{set } xfList. \pi_1 \ xf \notin \text{varDiffs}$
 and $\text{solHyp1}:\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1 \ uxf) \ 0 (\text{sol } s) = (\text{sol } s) (\pi_1 (\pi_2 \ uxf))$
 and $ygHyp:(y, g) \in \text{set } xfList$
 shows $(\text{sol } s[xfList \leftarrow uInput] \ 0)(\partial y) = g (\text{sol } s[xfList \leftarrow uInput] \ 0)$
 using *assms* apply (*induct xfList uInput rule: list-induct2', simp, simp, simp, clarify*)
 by (*rule inductive-to-sol-zero-its-dvars, simp-all*)

lemma *inductive-to-sol-greater-than-zero-its-dvars*:
 assumes $\text{lengthHyp}:\text{length } ((y, g) \# xfs) = \text{length } (v \# vs)$
 and $\text{distHyp}:\text{distinct } (\text{map } \pi_1 ((y, g) \# xfs))$
 and $\text{varHyp}:\forall xf \in \text{set } ((y, g) \# xfs). \pi_1 \ xf \notin \text{varDiffs}$
 and $\text{indHyp}:(u, x, f) \in \text{set } (vs \otimes xfs) \implies (s[xfs \leftarrow vs] t)(\partial x) = \text{vderiv-of } (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t$
 and $\text{disjHyp}:(v, y, g) = (u, x, f) \vee (u, x, f) \in \text{set } (vs \otimes xfs)$ and $tHyp:t > 0$
 shows $(s[(y, g) \# xfs \leftarrow v \# vs] t) (\partial x) = \text{vderiv-of } (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t$
 proof –
 let $?lhs = ((s[xfs \leftarrow vs] t)(y := v \ t \ s, \partial y := \text{vderiv-of } (\lambda r. v \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t)) (\partial x)$
 let $?rhs = \text{vderiv-of } (\lambda r. u \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t$
 have $(s[(y, g) \# xfs \leftarrow v \# vs] t) (\partial x) = ?lhs$ using $tHyp$ by *simp*
 also have $\text{vderiv-of } (\lambda r. u \ r \ s) \ \{0 < .. < 2 *_{\mathbb{R}} t\} \ t = ?rhs$ by *simp*
 ultimately have $\text{obs}:\text{?thesis} = (?lhs = ?rhs)$ by *simp*
 from disjHyp have $?lhs = ?rhs$
 proof
 assume $uxfEq:(v, y, g) = (u, x, f)$
 then have $?lhs = \text{vderiv-of } (\lambda r. u \ r \ s) \ \{0 < .. < (2 \cdot t)\} \ t$ by *simp*

also have $vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0 < .. < (2 \cdot t)\}\ t = ?rhs$ using $uxfEq$ by $simp$
 ultimately show $?lhs = ?rhs$ by $simp$
 next
 assume $sygTail:(u, x, f) \in set\ (vs \otimes xfs)$
 from this have $y \neq x$ using $distHyp\ set\text{-}zip\text{-}left\text{-}rightD$ by $force$
 hence $\partial\ x \neq \partial\ y$ by $(simp\ add:\ vdiff\text{-}def)$
 have $y \neq \partial\ x$ using $varHyp$ using $vdiff\text{-}invarDiffs$ by $auto$
 then have $?lhs = (s[xfs \leftarrow vs]\ t)\ (\partial\ x)$ using $\langle y \neq \partial\ x \rangle$ and $\langle \partial\ x \neq \partial\ y \rangle$ by $simp$
 also have $(s[xfs \leftarrow vs]\ t)\ (\partial\ x) = ?rhs$ using $indHyp\ sygTail$ by $simp$
 ultimately show $?lhs = ?rhs$ by $simp$
 qed
 from this and obs show $?thesis$ by $simp$
 qed

lemma *to-sol-greater-than-zero-its-dvars*:
 assumes $distinctHyp:distinct\ (map\ \pi_1\ xfList)$
 and $lengthHyp:length\ xfList = length\ uInput$
 and $varsHyp:\forall\ xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
 and $uxfHyp:(u, x, f) \in set\ (uInput \otimes xfList)$ and $tHyp:t > 0$
 shows $(s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0 < .. < (2 \cdot_R\ t)\}\ t$
 using $assms$ apply $(induct\ xfList\ uInput\ rule:\ list\text{-}induct2',\ simp,\ simp,\ simp,\ clarify)$
 by $(rule\text{-}tac\ f=f\ in\ inductive\text{-}to\text{-}sol\text{-}greater\text{-}than\text{-}zero\text{-}its\text{-}dvars,\ auto)$

dInv preliminaries

Here, we introduce syntactic notation to talk about differential invariants.

no-notation *Antidomain-Semiring.antidomain-left-monoid-class.am-add-op* (**infixl** \oplus 65)

no-notation *Dioid.times-class.opp-mult* (**infixl** \odot 70)

no-notation *Lattices.inf-class.inf* (**infixl** \sqcap 70)

no-notation *Lattices.sup-class.sup* (**infixl** \sqcup 65)

datatype $trms = Const\ real\ (t_C - [54]\ 70) \mid Var\ string\ (t_V - [54]\ 70) \mid$
 $Mns\ trms\ (\ominus - [54]\ 65) \mid Sum\ trms\ trms\ (\mathbf{infixl}\ \oplus\ 65) \mid$
 $Mult\ trms\ trms\ (\mathbf{infixl}\ \odot\ 68)$

primrec $tval :: trms \Rightarrow (real\ store \Rightarrow real)\ ((1\ \ll -\ \ll_t))$ **where**

$\ll_{t_C}\ r\ \ll_t = (\lambda\ s.\ r)$
 $\ll_{t_V}\ x\ \ll_t = (\lambda\ s.\ s\ x)$
 $\ll_{\ominus\ \vartheta}\ \ll_t = (\lambda\ s.\ -\ (\ll_{\vartheta}\ \ll_t)\ s)$
 $\ll_{\vartheta\ \oplus\ \eta}\ \ll_t = (\lambda\ s.\ (\ll_{\vartheta}\ \ll_t)\ s + (\ll_{\eta}\ \ll_t)\ s)$
 $\ll_{\vartheta\ \odot\ \eta}\ \ll_t = (\lambda\ s.\ (\ll_{\vartheta}\ \ll_t)\ s \cdot (\ll_{\eta}\ \ll_t)\ s)$

datatype $props = Eq\ trms\ trms\ (\mathbf{infixr}\ \doteq\ 60) \mid Less\ trms\ trms\ (\mathbf{infixr}\ \prec\ 62) \mid$
 $Leq\ trms\ trms\ (\mathbf{infixr}\ \preceq\ 61) \mid And\ props\ props\ (\mathbf{infixl}\ \sqcap\ 63) \mid$
 $Or\ props\ props\ (\mathbf{infixl}\ \sqcup\ 64)$

primrec $pval :: props \Rightarrow (real\ store \Rightarrow bool)\ ((1\ \ll -\ \ll_P))$ **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* (∂_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* (∂_P - [54] 70) **where**

$$\begin{aligned}
(\partial_P (\vartheta \doteq \eta)) &= ((\partial_t \vartheta) \doteq (\partial_t \eta)) | \\
(\partial_P (\vartheta \prec \eta)) &= ((\partial_t \vartheta) \preceq (\partial_t \eta)) | \\
(\partial_P (\vartheta \preceq \eta)) &= ((\partial_t \vartheta) \preceq (\partial_t \eta)) | \\
(\partial_P (\varphi \sqcap \psi)) &= (\partial_P \varphi) \sqcap (\partial_P \psi) | \\
(\partial_P (\varphi \sqcup \psi)) &= (\partial_P \varphi) \sqcap (\partial_P \psi)
\end{aligned}$$

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* ($\langle \cdot \rangle$ [54] 80) **where**

$$\begin{aligned}
\text{xtList } \langle t_C r \rangle &= t_C r | \\
\llbracket \langle t_V x \rangle &= t_V x | \\
\langle (y, \xi) \# \text{xtTail } \langle \text{Var } x \rangle &= (\text{if } x = y \text{ then } \xi \text{ else } \text{xtTail } \langle \text{Var } x \rangle) | \\
\text{xtList } \langle \ominus \vartheta \rangle &= \ominus (\text{xtList } \langle \vartheta \rangle) | \\
\text{xtList } \langle \vartheta \oplus \eta \rangle &= (\text{xtList } \langle \vartheta \rangle) \oplus (\text{xtList } \langle \eta \rangle) | \\
\text{xtList } \langle \vartheta \odot \eta \rangle &= (\text{xtList } \langle \vartheta \rangle) \odot (\text{xtList } \langle \eta \rangle)
\end{aligned}$$

proposition *substList-on-compl-of-varDiffs*:

assumes *trmVars* $\eta \subseteq (\text{UNIV} - \text{varDiffs})$

and *set* (*map* π_1 *xtList*) $\subseteq \text{varDiffs}$

shows *xtList* $\langle \eta \rangle = \eta$

using *assms* **apply** (*induction* η , *simp-all* *add*: *varDiffs-def*)

by (*induction* *xtList*, *auto*)

lemma *substList-help1*: *set* (*map* π_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 *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* :: *props* \Rightarrow *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*: $((\cdot) \ a \ \text{has-vector-derivative} \ a) \ (\text{at } x \ \text{within } T)$
by (*auto intro: derivative-eq-intros*)

lemma [*poly-deriv*]: $((\cdot) \ a \ \text{has-derivative} \ (\lambda x. x *_{\mathbb{R}} a)) \ (\text{at } x \ \text{within } T)$
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*(15))
by (*auto intro: derivative-eq-intros*)

lemma *quadratic-monomial-derivative2*:
 $((\lambda t::\text{real}. a \cdot t^2 / 2) \ \text{has-derivative} \ (\lambda t. a \cdot x \cdot t)) \ (\text{at } x \ \text{within } T)$
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*]: $((\lambda t. a \cdot t^2 / 2) \ \text{has-vderiv-on} \ (\cdot) \ a) \ T$
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*]:
 $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \ \text{has-vderiv-on} \ (\lambda t. a \cdot t + v)) \ T$
apply(*rule-tac* $f' = \lambda x. a \cdot x + v$ **and** $g'1 = \lambda x. 0$ **in** *derivative-intros*(191))
apply(*rule-tac* $f'1 = \lambda x. a \cdot x$ **and** $g'1 = \lambda x. v$ **in** *derivative-intros*(191))
using *poly-deriv*(2) **by** (*auto intro: derivative-intros*)

lemma [*poly-deriv*]:
 $t \in T \implies ((\lambda \tau. a \cdot \tau^2 / 2 + v \cdot \tau + x) \ \text{has-derivative} \ (\lambda x. x *_{\mathbb{R}} (a \cdot t + v)))$
 $(\text{at } t \ \text{within } T)$
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 $\exists f. ((\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \ \text{has-vderiv-on} \ f) \ \{0 <..< 2 \cdot t\}$
using *galilean-position* **by** *blast*
hence $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \ \text{has-vderiv-on} \ ?f) \ \{0 <..< 2 \cdot t\}$
unfolding *vderiv-of-def* **by** (*metis* (*mono-tags, lifting*) *someI-ex*)
also have $((\lambda t. a \cdot t^2 / 2 + v \cdot t + x) \ \text{has-vderiv-on} \ (\lambda t. a \cdot t + v)) \ \{0 <..< 2 \cdot t\}$
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  $\langle t \in \{0 < .. < 2 \cdot t\} \rangle$  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(191))
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 = 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  $\exists f. ((\lambda r. a \cdot r + v) \text{ has-vderiv-on } f) \{0 < .. < 2 \cdot t\}$ 
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 (( $\lambda r. a \cdot r + v$ ) has-vderiv-on ( $\lambda t. a$ ))  $\{0 < .. < 2 \cdot t\}$ 
using galilean-velocity by simp
ultimately show (vderiv-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]:
(( $\lambda t. v \cdot t - a \cdot t^2 / 2 + x$ ) has-vderiv-on ( $\lambda x. v - a \cdot x$ ))  $\{0..t\}$ 
apply(subgoal-tac (( $\lambda t. - a \cdot t^2 / 2 + v \cdot t + x$ ) has-vderiv-on ( $\lambda x. - a \cdot x + v$ ))  $\{0..t\}$ , simp)
by(rule galilean-transform)

```

```

lemma [galilean-transform-eq]: $t > 0 \implies \text{vderiv-of } (\lambda t. v \cdot t - a \cdot t^2 / 2 + x) \{0 < .. < 2 \cdot t\} t = v - a \cdot t$ 
apply(subgoal-tac vderiv-of ( $\lambda t. - a \cdot t^2 / 2 + v \cdot t + x$ )  $\{0 < .. < 2 \cdot t\} t = - a \cdot t + v$ , simp)
by(rule galilean-transform-eq)

```

```

lemma [galilean-transform]:
  (( $\lambda t. v - a \cdot t$ ) has-vderiv-on ( $\lambda x. - a$ )) {0..t}
apply(subgoal-tac (( $\lambda t. - a \cdot t + v$ ) has-vderiv-on ( $\lambda x. - a$ )) {0..t}, simp)
by(rule galilean-transform)

lemma [galilean-transform-eq]:  $t > 0 \implies \text{vderiv-of } (\lambda r. v - a \cdot r) \{0 < .. < 2 \cdot t\}$ 
 $t = - a$ 
apply(subgoal-tac vderiv-of ( $\lambda t. - a \cdot t + v$ ) {0 < .. < 2 · t}  $t = - a$ , 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

```

0.20.3 Phase Space Relational Semantics

```

definition solvesStoreIVP :: (real  $\Rightarrow$  real store)  $\Rightarrow$  (string  $\times$  (real store  $\Rightarrow$  real))
list  $\Rightarrow$ 
real store  $\Rightarrow$  bool
((- solvesTheStoreIVP - withInitState -) [70, 70, 70] 68) where
solvesStoreIVP  $\varphi_S$  xfList s  $\equiv$ 
  — F sends vdiffs-in-list to derivs.
  ( $\forall t \geq 0. (\forall xf \in \text{set } xfList. \varphi_S t (\partial (\pi_1 xf)) = \pi_2 xf (\varphi_S t)) \wedge$ 
  — F preserves the rest of the variables and F sends derivs of constants to 0.
  ( $\forall y. (y \notin (\pi_1(\text{set } xfList)) \cup \text{varDiffs} \longrightarrow \varphi_S t y = s y) \wedge$ 
    ( $y \notin (\pi_1(\text{set } xfList)) \longrightarrow \varphi_S t (\partial y) = 0$ ))  $\wedge$ 
  — F solves the induced IVP.
  ( $\forall xf \in \text{set } xfList. ((\lambda t. \varphi_S t (\pi_1 xf)) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 xf) (\varphi_S t))) \{0..t\}$ 
    UNIV  $\wedge$ 
     $\varphi_S 0 (\pi_1 xf) = s(\pi_1 xf))$ )

lemma solves-store-ivpI:
assumes  $\forall t \geq 0. \forall xf \in \text{set } xfList. (\varphi_S t (\partial (\pi_1 xf))) = (\pi_2 xf) (\varphi_S t)$ 
and  $\forall t \geq 0. \forall y. y \notin (\pi_1(\text{set } xfList)) \cup \text{varDiffs} \longrightarrow \varphi_S t y = s y$ 
and  $\forall t \geq 0. \forall y. y \notin (\pi_1(\text{set } xfList)) \longrightarrow \varphi_S t (\partial y) = 0$ 
and  $\forall t \geq 0. \forall xf \in \text{set } xfList. ((\lambda t. \varphi_S t (\pi_1 xf)) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 xf) (\varphi_S t))) \{0..t\}$ 
UNIV
and  $\forall xf \in \text{set } xfList. \varphi_S 0 (\pi_1 xf) = s(\pi_1 xf)$ 
shows  $\varphi_S \text{ solvesTheStoreIVP } xfList \text{ withInitState } s$ 
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 φ_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 φ_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** $x \in (\pi_1(\text{set } \text{xfList})) \implies \varphi_S 0 x = s x$
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 φ_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 φ_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 φ_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 φ_S *solvesTheStoreIVP* *xfList* *withInitState* *s*
and $t \geq 0$

and $xf \in \text{set } xfList$
shows $((\lambda t. \varphi_S t (\pi_1 xf)) \text{ solves-ode } (\lambda t. \lambda r. (\pi_2 xf) (\varphi_S t))) \{0..t\} \text{ UNIV}$
using *assms solves-store-ivpE(4)* **by** *simp*

lemma [*solves-store-ivpD*]:
assumes $\varphi_S \text{ solvesTheStoreIVP } xfList \text{ withInitState } s$
and $(x, f) \in \text{set } xfList$
shows $\varphi_S 0 x = s x$
using *assms solves-store-ivpE(5)* **by** *fastforce*

lemma [*solves-store-ivpD*]:
assumes $\varphi_S \text{ solvesTheStoreIVP } xfList \text{ withInitState } s$
and $y \notin \text{varDiffs}$
shows $\varphi_S 0 y = s y$
using *assms solves-store-ivpE(6)* **by** *simp*

definition *guardDiffEqtn* :: $(\text{string} \times (\text{real store} \Rightarrow \text{real})) \text{ list} \Rightarrow (\text{real store} \text{ pred})$
 \Rightarrow
 $\text{real store rel } (\text{ODEsystem} - \text{with} - [70, 70] 61) \text{ where}$
 $\text{ODEsystem } xfList \text{ with } G = \{(s, \varphi_S t) \mid s t \varphi_S. t \geq 0 \wedge (\forall r \in \{0..t\}. G (\varphi_S r))$
 $\wedge \text{solvesStoreIVP } \varphi_S xfList s\}$

0.20.4 Derivation of Differential Dynamic Logic Rules

”Differential Weakening”

lemma *wlp-evol-guard*: $\text{Id} \subseteq \text{wp } (\text{ODEsystem } xfList \text{ with } G) \lceil G \rceil$
by (*simp add: rel-antidomain-kleene-algebra.fbox-def rel-ad-def guardDiffEqtn-def p2r-def, force*)

theorem *dWeakening*:
assumes *guardImpliesPost*: $\lceil G \rceil \subseteq \lceil Q \rceil$
shows $\text{PRE } P (\text{ODEsystem } xfList \text{ with } G) \text{ POST } Q$
using *assms and wlp-evol-guard by (metis (no-types, hide-lams) d-p2r order-trans p2r-subid rel-antidomain-kleene-algebra.fbox-iso)*

theorem *dW*: $\text{wp } (\text{ODEsystem } xfList \text{ with } G) \lceil Q \rceil = \text{wp } (\text{ODEsystem } xfList \text{ with } G) \lceil \lambda s. G s \longrightarrow Q s \rceil$
unfolding *rel-antidomain-kleene-algebra.fbox-def rel-ad-def guardDiffEqtn-def*
by (*simp add: relcomp.simps p2r-def, fastforce*)

”Differential Cut”

lemma *all-interval-guardDiffEqtn*:
assumes $\text{solvesStoreIVP } \varphi_S xfList s \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t$
shows $\forall r \in \{0..t\}. (s, \varphi_S r) \in (\text{ODEsystem } xfList \text{ with } G)$
unfolding *guardDiffEqtn-def* **using** *atLeastAtMost-iff* **apply** *clarsimp*
apply (*rule-tac x=r in exI, rule-tac x= φ_S in exI*) **using** *assms by simp*

lemma *condAfterEvol-remainsAlongEvol*:

assumes $\text{boxDiffC}:(s, s) \in \text{wp} (\text{ODEsystem } \text{xfList with } G) \lceil C \rceil$
and $\text{FisSol}:\text{solvesStoreIVP } \varphi_S \text{ xfList } s \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t$
shows $\forall r \in \{0..t\}. G (\varphi_S r) \wedge C (\varphi_S r)$
proof–
from boxDiffC **have** $\forall c. (s, c) \in (\text{ODEsystem } \text{xfList with } G) \longrightarrow C c$
by (*simp add: boxProgrPred-chrcrtn*)
also from FisSol **have** $\forall r \in \{0..t\}. (s, \varphi_S r) \in (\text{ODEsystem } \text{xfList with } G)$
using *all-interval-guarDiffEqtn* **by** *blast*
ultimately show *?thesis*
using FisSol *atLeastAtMost-iff guarDiffEqtn-def* **by** *fastforce*
qed

theorem *dCut*:
assumes $p\text{BoxDiffCut}:(\text{PRE } P (\text{ODEsystem } \text{xfList with } G) \text{ POST } C)$
assumes $p\text{BoxCutQ}:(\text{PRE } P (\text{ODEsystem } \text{xfList with } (\lambda s. G s \wedge C s)) \text{ POST } Q)$
shows $\text{PRE } P (\text{ODEsystem } \text{xfList with } G) \text{ POST } Q$
apply(*clarify*, *subgoal-tac a = b*) **defer**
proof(*metis d-p2r rdom-p2r-contents*, *simp*, *subst boxProgrPred-chrcrtn*, *clarify*)
fix $b y$ **assume** $(b, b) \in \lceil P \rceil$ **and** $(b, y) \in \text{ODEsystem } \text{xfList with } G$
then obtain $\varphi_S t$ **where** $:\text{solvesStoreIVP } \varphi_S \text{ xfList } b \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t \wedge \varphi_S t = y$
using *guarDiffEqtn-def* **by** *auto*
hence $\forall r \in \{0..t\}. (b, \varphi_S r) \in (\text{ODEsystem } \text{xfList with } G)$
using *all-interval-guarDiffEqtn* **by** *blast*
from this and $p\text{BoxDiffCut}$ **have** $\forall r \in \{0..t\}. C (\varphi_S r)$
using *boxProgrPred-chrcrtn* $\langle (b, b) \in \lceil P \rceil \rangle$ **by** (*metis (no-types, lifting) d-p2r subsetCE*)
then have $\forall r \in \{0..t\}. (b, \varphi_S r) \in (\text{ODEsystem } \text{xfList with } (\lambda s. G s \wedge C s))$
using \ast *all-interval-guarDiffEqtn* **by** (*metis (mono-tags, lifting)*)
from this and $p\text{BoxCutQ}$ **have** $\forall r \in \{0..t\}. Q (\varphi_S r)$
using *boxProgrPred-chrcrtn* $\langle (b, b) \in \lceil P \rceil \rangle$ **by** (*metis (no-types, lifting) d-p2r subsetCE*)
thus $Q y$ **using** \ast **by** *auto*
qed

theorem *dC*:
assumes $\text{Id} \subseteq \text{wp} (\text{ODEsystem } \text{xfList with } G) \lceil C \rceil$
shows $\text{wp} (\text{ODEsystem } \text{xfList with } G) \lceil Q \rceil = \text{wp} (\text{ODEsystem } \text{xfList with } (\lambda s. G s \wedge C s)) \lceil Q \rceil$
proof(*rule-tac f = \lambda x. wp x \lceil Q \rceil* **in** *HOL.arg-cong, safe*)
fix $a b$ **assume** $(a, b) \in \text{ODEsystem } \text{xfList with } G$
then obtain $\varphi_S t$ **where** $:\text{solvesStoreIVP } \varphi_S \text{ xfList } a \wedge (\forall r \in \{0..t\}. G (\varphi_S r)) \wedge 0 \leq t \wedge \varphi_S t = b$
using *guarDiffEqtn-def* **by** *auto*
hence $1:\forall r \in \{0..t\}. (a, \varphi_S r) \in \text{ODEsystem } \text{xfList with } G$
by (*meson all-interval-guarDiffEqtn*)
from this have $\forall r \in \{0..t\}. C (\varphi_S r)$ **using** *assms boxProgrPred-chrcrtn*
by (*metis IdI boxProgrPred-IsProp subset-antisym*)
thus $(a, b) \in \text{ODEsystem } \text{xfList with } (\lambda s. G s \wedge C s)$

```

    using * guarDiffEqtn-def by blast
next
  fix a b assume (a, b) ∈ ODEsystem xflist with (λs. G s ∧ C s)
  then show (a, b) ∈ ODEsystem xflist with G
  unfolding guarDiffEqtn-def by (clarsimp, rule-tac x=t in exI, rule-tac x=φS in
exI, simp)
qed

```

Solve Differential Equation

lemma *prelim-dSolve*:

```

assumes solHyp:(λt. sol s[xflist←uInput] t) solvesTheStoreIVP xflist withInit-
State s
and uniqHyp:∀ X. solvesStoreIVP X xflist s ⟶ (∀ t ≥ 0. (sol s[xflist←uInput]
t) = X t)
and diffAssgn: ∀ t ≥ 0. G (sol s[xflist←uInput] t) ⟶ Q (sol s[xflist←uInput] t)
shows ∀ c. (s, c) ∈ (ODEsystem xflist with G) ⟶ Q c
proof(clarify)
fix c assume (s, c) ∈ (ODEsystem xflist with G)
from this obtain t::real and φS::real ⇒ real store
where FHyp:t ≥ 0 ∧ φS t = c ∧ solvesStoreIVP φS xflist s ∧ (∀ r ∈ {0..t}. G
(φS r))
using guarDiffEqtn-def by auto
from this and uniqHyp have (sol s[xflist←uInput] t) = φS t by blast
then have cHyp:c = (sol s[xflist←uInput] t) using FHyp by simp
from this have G (sol s[xflist←uInput] t) using FHyp by force
then show Q c using diffAssgn FHyp cHyp by auto
qed

```

theorem *dS*:

```

assumes solHyp:∀ s. solvesStoreIVP (λt. sol s[xflist←uInput] t) xflist s
and uniqHyp:∀ s X. solvesStoreIVP X xflist s ⟶ (∀ t ≥ 0. (sol s[xflist←uInput]
t) = X t)
shows wp (ODEsystem xflist with G) [Q] =
  [λ s. ∀ t ≥ 0. (∀ r ∈ {0..t}. G (sol s[xflist←uInput] r)) ⟶ Q (sol s[xflist←uInput]
t)]
apply(simp add: p2r-def, rule subset-antisym)
unfolding guarDiffEqtn-def rel-antidomain-kleene-algebra.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←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=λt. sol x[xflist←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:∀ s. solvesStoreIVP (λt. sol s[xflist←uInput] t) xflist s
and uniqHyp:∀ s. ∀ X. solvesStoreIVP X xflist s ⟶ (∀ t ≥ 0. (sol s[xflist←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*: $\forall s\ g. \forall xf \in \text{set } xfList. \pi_2\ xf\ (\text{override-on } s\ g\ \text{varDiffs}) = \pi_2\ xf\ s$
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1\ xf \notin \text{varDiffs}$
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *solHyp1*: $\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$
and *solHyp2*: $\forall t \geq 0. \forall xf \in \text{set } xfList. ((\lambda \tau. (sol\ s[xfList \leftarrow uInput]\ \tau)\ (\pi_1\ xf))\ has-vderiv-on\ (\lambda \tau. (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau)))\ \{0..t\}$
shows $\forall t \geq 0. \forall xf \in \text{set } xfList. (sol\ s[xfList \leftarrow uInput]\ t)\ (\partial\ (\pi_1\ xf)) = (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ t)$
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 $\forall x. x \notin (\pi_1(\text{set } xfList)) \longrightarrow (sol\ s[xfList \leftarrow uInput]\ t)\ (\partial\ x) = 0$
using *varsHyp* **apply**(*induct* *xfList* *uInput* *rule: list-induct2'*)
apply(*simp-all* *add: override-on-def varDiffs-def vdiff-def*)
by *clarsimp*

lemma *conds4InitState*:

assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length* *xfList* = *length* *uInput*
and *varsHyp*: $\forall xf \in \text{set } xfList. \pi_1\ xf \notin \text{varDiffs}$
and *solHyp1*: $\forall uxf \in \text{set } (uInput \otimes xfList). (\pi_1\ uxf)\ 0\ (sol\ s) = (sol\ s)\ (\pi_1\ (\pi_2\ uxf))$
and *xfHyp*: $(x, f) \in \text{set } xfList$
shows $(sol\ s[xfList \leftarrow uInput]\ 0)\ x = s\ x$
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*: $(sol\ s)\ x = s\ x$ **using** *override-on-def xfHyp* **by** *auto*
from *uxfHyp* **and** *solHyp1* **have** $u\ 0\ (sol\ s) = (sol\ s)\ x$ **by** *fastforce*
also **have** $(sol\ s[xfList \leftarrow uInput]\ 0)\ x = u\ 0\ (sol\ s)$
using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*
ultimately show $(sol\ s[xfList \leftarrow uInput]\ 0)\ x = s\ x$ **using** *toZeroHyp* **by** *simp*
qed

lemma *conds4RestOfStrings*:

assumes $x \notin (\pi_1(\text{set } xfList)) \cup \text{varDiffs}$
shows $(sol\ s[xfList \leftarrow uInput]\ t)\ x = s\ x$
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  $\in$  set xfList.  $\pi_1$  xf  $\notin$  varDiffs
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.
(( $\lambda t$ . (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}
shows solvesStoreIVP ( $\lambda t$ . (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  $\in$  set xfList.  $\pi_2$  xf (override-on s g varDiffs) =  $\pi_2$  xf s
and distinctHyp:distinct (map  $\pi_1$  xfList)
and lengthHyp:length xfList = length uInput
and varsHyp: $\forall$  xf  $\in$  set xfList.  $\pi_1$  xf  $\notin$  varDiffs
and solHyp1: $\forall$  s.  $\forall$  uxf  $\in$  set (uInput  $\otimes$  xfList). ( $\pi_1$  uxf) 0 (sol s) = (sol s) ( $\pi_1$  ( $\pi_2$  uxf))
and solHyp2: $\forall$  s.  $\forall$  t  $\geq$  0.  $\forall$  xf  $\in$  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) ( $\lambda(t, (r::\text{real})). f$  ( $\varphi_s$  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* φ_s *xfList* *a*
shows $\varphi_s \ 0 = \text{override-on } a \ (\varphi_s \ 0) \ \text{varDiffs}$
apply(*rule ext*, *subgoal-tac* $x \notin \text{varDiffs} \longrightarrow \varphi_s \ 0 \ x = a \ x$)
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*: $\forall \ xf \in \text{set } \text{xfList}. \text{continuous-on } (\{0..t\} \times \text{UNIV})$
 $(\lambda(t, (r::\text{real})). (\pi_2 \ xf) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t))$
and *eqDerivs*: $\forall \ xf \in \text{set } \text{xfList}. \forall \ \tau \in \{0..t\}. (\pi_2 \ xf) \ (\varphi_s \ \tau) = (\pi_2 \ xf) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau)$
and *Fsolves*: *solvesStoreIVP* φ_s *xfList* *s*
and *solHyp*: *solvesStoreIVP* $(\lambda \ \tau. (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau))$ *xfList* *s*
shows $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) = \varphi_s \ t$
proof
fix *x::string* **show** $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ x = \varphi_s \ t \ x$
proof(*cases* $x \in (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs}$)
case *False*
then have *notInVars*: $x \notin (\pi_1(\text{set } \text{xfList})) \cup \text{varDiffs}$ **by** *simp*
from *solHyp* **have** $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ x = s \ x$
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 $(\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ t) \ x = \varphi_s \ t \ x$ **by** *simp*
next case *True*
then have $x \in (\pi_1(\text{set } \text{xfList})) \vee x \in \text{varDiffs}$ **by** *simp*
from this show *?thesis*
proof
assume $x \in (\pi_1(\text{set } \text{xfList}))$
from this obtain *f* **where** *xfHyp*: $(x, f) \in \text{set } \text{xfList}$ **by** *fastforce*

then have *expand1*: $\forall \ xf \in \text{set } \text{xfList}. ((\lambda \tau. \varphi_s \ \tau \ (\pi_1 \ xf)) \text{ solves-ode } (\lambda \tau \ r. (\pi_2 \ xf) \ (\varphi_s \ \tau))) \{0..t\} \ \text{UNIV} \wedge \varphi_s \ 0 \ (\pi_1 \ xf) = s \ (\pi_1 \ xf)$
using *Fsolves tHyp* **by** (*simp add: solvesStoreIVP-def*)
hence *expand2*: $\forall \ xf \in \text{set } \text{xfList}. \forall \ \tau \in \{0..t\}. ((\lambda r. \varphi_s \ r \ (\pi_1 \ xf)) \text{ has-vector-derivative } (\lambda r. (\pi_2 \ xf) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau)) \ \tau) \text{ (at } \tau \text{ within } \{0..t\}))$
using *eqDerivs* **by** (*simp add: solves-ode-def has-vderiv-on-def*)

then have $\forall \ xf \in \text{set } \text{xfList}. ((\lambda \tau. \varphi_s \ \tau \ (\pi_1 \ xf)) \text{ solves-ode } (\lambda \tau \ r. (\pi_2 \ xf) \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau))) \{0..t\} \ \text{UNIV} \wedge \varphi_s \ 0 \ (\pi_1 \ xf) = s \ (\pi_1 \ xf)$
by (*simp add: has-vderiv-on-def solves-ode-def expand1 expand2*)
then have $1: ((\lambda \tau. \varphi_s \ \tau \ x) \text{ solves-ode } (\lambda \tau \ r. f \ (\text{sol } s[\text{xfList} \leftarrow \text{uInput}] \ \tau))) \{0..t\}$


```

UNIV  $\wedge$ 
   $\varphi_s$  0  $x = s$   $x$  using  $xfHyp$  by fastforce

  from  $solHyp$  and  $xfHyp$  have 2:  $((\lambda \tau. (sol\ s[xfList \leftarrow uInput]\ \tau)\ x)\ solves\ ode$ 
     $(\lambda \tau\ r. f\ (sol\ s[xfList \leftarrow uInput]\ \tau)))\ \{0..t\}\ UNIV \wedge (sol\ s[xfList \leftarrow uInput]\ 0)$ 
 $x = s\ x$ 
    using solvesStoreIVP-def  $tHyp$  by fastforce

  from  $tHyp$  and  $contHyp$  have  $\forall\ xf \in set\ xfList. unique\ on\ bounded\ closed\ 0$ 
 $\{0..t\}\ (s\ (\pi_1\ xf))$ 
     $(\lambda \tau\ r. (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ \tau))\ UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$ 

  apply(clarify) apply(rule\ conds4UniqSol) by(auto)
  from this have 3:  $unique\ on\ bounded\ closed\ 0\ \{0..t\}\ (s\ x)\ (\lambda \tau\ r. f\ (sol$ 
 $s[xfList \leftarrow uInput]\ \tau))$ 
     $UNIV\ (if\ t = 0\ then\ 1\ else\ 1/(t+1))$  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  $x \in varDiffs$ 
  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  $y \in set\ (map\ \pi_1\ xfList)$ )
  case True
    then obtain  $f$  where  $xfHyp:(y, f) \in set\ xfList$  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  $contHyp: \forall\ s. \forall\ t \geq 0. \forall\ xf \in set\ xfList. continuous\ on\ (\{0..t\} \times UNIV)$ 
 $(\lambda(t, (r::real)). (\pi_2\ xf)\ (sol\ s[xfList \leftarrow uInput]\ t))$ 
and  $solHyp: \forall\ s. solvesStoreIVP\ (\lambda\ t. (sol\ s[xfList \leftarrow uInput]\ t))\ xfList\ s$ 
and  $uniqHyp: \forall\ s. \forall\ \varphi_s. \varphi_s\ solvesTheStoreIVP\ xfList\ withInitState\ s \longrightarrow$ 

```

$(\forall t \geq 0. \forall xf \in \text{set } xfList. \forall r \in \{0..t\}. (\pi_2 xf) (\varphi_s r) = (\pi_2 xf) (sol\ s[xfList \leftarrow uInput]\ r))$
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** $\varphi_s::\text{real} \Rightarrow \text{real store}$ **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 $\forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$
 $(\lambda(t, (r::\text{real})). (\pi_2 xf) (sol\ s[xfList \leftarrow uInput]\ t))$ **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*: *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*: $\forall s. \forall uxf \in \text{set } (uInput \otimes xfList). \pi_1\ uxf\ 0\ (sol\ s) = sol\ s\ (\pi_1\ (\pi_2\ uxf))$
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 *contHyp*: $\forall s. \forall t \geq 0. \forall xf \in \text{set } xfList. \text{continuous-on } (\{0..t\} \times UNIV)$
 $(\lambda(t, (r::\text{real})). (\pi_2\ xf) (sol\ s[xfList \leftarrow uInput]\ t))$
and *uniqHyp*: $\forall s. \forall \varphi_s. \varphi_s\ \text{solvesTheStoreIVP}\ xfList\ \text{withInitState}\ s \longrightarrow$
 $(\forall t \geq 0. \forall xf \in \text{set } xfList. \forall r \in \{0..t\}. (\pi_2\ xf) (\varphi_s\ r) = (\pi_2\ xf) (sol\ s[xfList \leftarrow uInput]\ r))$
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** *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*::*real* \Rightarrow *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::real
assume tHyp:  $0 \leq t$  and xHyp:  $(x, f) \in \text{set } xfList$  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  $\forall t \geq 0. \forall r \in \{0..t\}. \forall x f \in \text{set } xfList. (F r (\partial (\pi_1 x f))) = (\pi_2 x f) (F r)$ 
using assms by auto
from this rHyp and xHyp 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 * rHyp by auto
qed

```

lemma derivationLemma-baseCase:

```

fixes F::real  $\Rightarrow$  real store
assumes solves:solvesStoreIVP F xfList a
shows  $\forall x \in (\text{UNIV} - \text{varDiffs}). \forall t \geq 0. \forall r \in \{0..t\}. ((\lambda t. F t x) \text{ has-vector-derivative } F r (\partial x)) \text{ (at } r \text{ within } \{0..t\})$ 
proof
fix x
assume  $x \in \text{UNIV} - \text{varDiffs}$ 
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  $\forall t \geq 0. \forall r \in \{0..t\}. \forall x f \in \text{set } xfList. ((\lambda t. F t (\pi_1 x f)) \text{ has-vector-derivative } F r (\partial (\pi_1 x f))) \text{ (at } r \text{ within } \{0..t\})$ 
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* $F \text{ xflist } a$
and $tHyp:t \geq 0$
and $termVarsHyp:\forall x \in trmVars \eta. x \in (UNIV - varDiffs)$
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* η)
case (*Const* r)
then show *?case* **by** *simp*
next
case (*Var* y)
then have $yHyp:y \in UNIV - varDiffs$ **by** *auto*
from *this tHyp* **and** *assms(1)* **show** *?case*
using *derivationLemma-baseCase* **by** *auto*
next
case (*Mns* η)
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(25)*)
by (*simp-all add: has-field-derivative-iff-has-vector-derivative*)
qed

lemma *diff-subst-prprty-4terms*:
assumes *solves*: $\forall xf \in set \text{ xflist}. F t (\partial (\pi_1 xf)) = \pi_2 xf (F t)$
and $tHyp:(t::real) \geq 0$
and $listsHyp:\text{map } \pi_2 \text{ xflist} = \text{map } tval \text{ uInput}$
and $termVarsHyp:trmVars \eta \subseteq (UNIV - varDiffs)$
shows $\llbracket \partial_t \eta \rrbracket_t (F t) = \llbracket ((\text{map } (vdiff \circ \pi_1) \text{ xflist}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t (F t)$

```

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  $x::string$  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-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-cross-vdiff-on-non-occurring-var by (metis(no-types, lifting) List.map.compositionality)
          thus ?thesis by simp
        qed
      qed
    qed

```

```

lemma eqInVars-impl-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-all)

```

```

lemma non-empty-funList-implies-non-empty-trmList:

```

shows $\forall \text{ list}. (x, f) \in \text{set list} \wedge \text{map } \pi_2 \text{ list} = \text{map tval tList} \longrightarrow (\exists \vartheta. \llbracket \vartheta \rrbracket_t = f \wedge \vartheta \in \text{set tList})$
by (*induction tList, auto*)

lemma *dInvForTrms-prelim*:

assumes *substHyp*:

$\forall \text{ st}. G \text{ st} \longrightarrow (\forall \text{ str}. \text{str} \notin (\pi_1(\text{set xfList})) \longrightarrow \text{st } (\partial \text{ str}) = 0) \longrightarrow$

$\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t \text{ st} = 0$

and *termVarsHyp*: $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$

and *listsHyp*: $\text{map } \pi_2 \text{ xfList} = \text{map tval uInput}$

shows $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (\text{ODEsystem xfList with } G) \longrightarrow \llbracket \eta \rrbracket_t c = 0)$

proof (*clarify*)

fix *c* **assume** *aHyp*: $\llbracket \eta \rrbracket_t a = 0$ **and** *cHyp*: $(a, c) \in \text{ODEsystem xfList with } G$

from this obtain *t::real* **and** *F::real* \Rightarrow *real store*

where *tcHyp*: $t \geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ xfList } a \wedge (\forall r \in \{0..t\}. G (F r))$

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 *obs1*: $\llbracket \eta \rrbracket_t (F 0) = 0$ **using** *aHyp* **by** *simp*

from *tcHyp* **have** *obs2*: $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative}$

$\llbracket \partial_t \eta \rrbracket_t (F r))$ (at *r* within $\{0..t\}$) **using** *derivationLemma termVarsHyp* **by** *blast*

have $\forall r \in \{0..t\}. \forall \text{ xf} \in \text{set xfList}. F r (\partial (\pi_1 \text{ xf})) = \pi_2 \text{ xf } (F r)$

using *tcHyp solves-store-ivpD(3)* **by** *fastforce*

hence $\forall r \in \{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) = \llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t (F r)$

using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*

also from *substHyp* **have** $\forall r \in \{0..t\}. \llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t (F r) = 0$

using *solves-store-ivpD(2)* *tcHyp* **by** *fastforce*

ultimately have $\forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } 0)$ (at *r* within $\{0..t\}$)

using *obs2* **by** *auto*

from this and *tcHyp* **have** $\forall s \in \{0..t\}. ((\lambda x. \llbracket \eta \rrbracket_t (F x)) \text{ has-derivative } (\lambda x. x *_R 0))$

(at *s* within $\{0..t\}$) **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 $\forall \text{ st}. G \text{ st} \longrightarrow (\forall \text{ str}. \text{str} \notin (\pi_1(\text{set xfList})) \longrightarrow \text{st } (\partial \text{ str}) = 0) \longrightarrow$

$\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t \eta \rangle \rrbracket_t \text{ st} = 0$

and *termVarsHyp*: $\text{trmVars } \eta \subseteq (\text{UNIV} - \text{varDiffs})$

and *listsHyp*: $\text{map } \pi_2 \text{ xfList} = \text{map tval uInput}$

and *eta-f*: $f = \llbracket \eta \rrbracket_t$

shows *PRE* $(\lambda s. f s = 0)$ (*ODEsystem xfList with* *G*) *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 (metis (full-types) d-p2r rdom-p2r-contents)
have  $\llbracket \eta \rrbracket_t a = 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t c = 0)$ 
using assms dInvForTrms-prelim by metis
from this and aHyp have  $\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t c = 0$ 
by blast
thus  $(a, b) \in wp\ (ODEsystem\ xfList\ with\ G)\ [\lambda s. \llbracket \eta \rrbracket_t s = 0]$ 
using aHyp by (simp add: boxProgrPred-chrctrzn)
qed

```

```

lemma diff-subst-prprty-4props:
assumes solves:  $\forall xf \in set\ xfList. F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$ 
and tHyp:  $t \geq 0$ 
and listsHyp:  $map\ \pi_2\ xfList = map\ tval\ uInput$ 
and propVarsHyp:  $propVars\ \varphi \subseteq (UNIV - varDiffs)$ 
shows  $\llbracket \partial_P\ \varphi \rrbracket_P\ (F\ t) = \llbracket ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput) \upharpoonright \partial_P\ \varphi \rrbracket_P\ (F\ t)$ 
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:
 $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1 \downarrow set\ xfList)) \longrightarrow st\ (\partial\ str) = 0 \longrightarrow$ 
 $\llbracket ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput) \langle \partial_t\ \eta \rangle \rrbracket_t\ st \geq 0$ 
and termVarsHyp:  $trmVars\ \eta \subseteq (UNIV - varDiffs)$ 
and listsHyp:  $map\ \pi_2\ xfList = map\ tval\ uInput$ 
shows  $\llbracket \eta \rrbracket_t a > 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t c > 0)$ 
and  $\llbracket \eta \rrbracket_t a \geq 0 \longrightarrow (\forall c. (a, c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow \llbracket \eta \rrbracket_t c \geq 0)$ 
proof(clarify)
fix c assume  $aHyp: \llbracket \eta \rrbracket_t a > 0$  and  $cHyp: (a, c) \in ODEsystem\ xfList\ with\ G$ 
from this obtain t::real and F::real  $\Rightarrow$  real store
where  $tcHyp: t \geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a \wedge (\forall r \in \{0..t\}. G\ (F\ r))$ 

```

```

using guarDiffEqtn-def by auto
then have  $\forall x. x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$  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  $obs1: \llbracket \eta \rrbracket_t (F\ 0) > 0$  using aHyp tcHyp by simp
from tcHyp have  $obs2: \forall r \in \{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F\ s))\ has\_vector\_derivative\ \langle \partial_t\ \eta \rangle_t\ (F\ r))$ 
  (at r within  $\{0..t\}$ ) using derivationLemma termVarsHyp by blast
have  $(\forall t \geq 0. \forall xf \in set\ xfList. F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$ 
using tcHyp solves-store-ivpD(3) by blast
hence  $\forall r \in \{0..t\}. \llbracket \partial_t\ \eta \rrbracket_t (F\ r) = \llbracket ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput) \langle \partial_t\ \eta \rangle \rrbracket_t\ (F\ r)$ 
using diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp by fastforce
also from substHyp have  $\forall r \in \{0..t\}. \llbracket ((map\ (vdiff\ \circ\ \pi_1)\ xfList) \otimes uInput) \langle \partial_t\ \eta \rangle \rrbracket_t$ 

```

$\eta\rangle_t (F r) \geq 0$
using *solves-store-ivpD(2) tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $\ast:\forall r\in\{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) \geq 0$ **by** (*simp*)
from *obs2* **and** *tcHyp* **have** $\forall r\in\{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-derivative } (\lambda x. x \ast_R (\llbracket \partial_t \eta \rrbracket_t (F r))))$ (*at r within \{0..t\}*) **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 *mvt-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 \ast *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $\llbracket \eta \rrbracket_t c > 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(45)
not-le)

next

show $0 \leq \llbracket \eta \rrbracket_t a \longrightarrow (\forall c. (a, c) \in \text{ODEsystem } \text{xfList with } G \longrightarrow 0 \leq \llbracket \eta \rrbracket_t c)$
proof(*clarify*)
fix c **assume** $a\text{Hyp}:\llbracket \eta \rrbracket_t a \geq 0$ **and** $c\text{Hyp}:(a, c) \in \text{ODEsystem } \text{xfList with } G$
from this obtain $t::\text{real}$ **and** $F::\text{real} \Rightarrow \text{real store}$
where $tcHyp:t\geq 0 \wedge F t = c \wedge \text{solvesStoreIVP } F \text{ xfList } a \wedge (\forall r\in\{0..t\}. G (F r))$

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 $\text{obs1}:\llbracket \eta \rrbracket_t (F 0) \geq 0$ **using** $a\text{Hyp}$ *tcHyp* **by** *simp*
from *tcHyp* **have** $\text{obs2}:\forall r\in\{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-vector-derivative } \llbracket \partial_t \eta \rrbracket_t (F r))$ (*at r within \{0..t\}*) **using** *derivationLemma termVarsHyp* **by** *blast*
have $(\forall t\geq 0. \forall \text{xf} \in \text{set } \text{xfList}. F t (\partial (\pi_1 \text{xf})) = \pi_2 \text{xf} (F t))$
using *tcHyp solves-store-ivpD(3)* **by** *blast*
from this and *tcHyp* **have** $\forall r\in\{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) =$
 $\llbracket ((\text{map } (v\text{diff} \circ \pi_1) \text{xfList}) \otimes u\text{Input}) \langle \partial_t \eta \rangle \rrbracket_t (F r)$
using *diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r\in\{0..t\}. \llbracket ((\text{map } (v\text{diff} \circ \pi_1) \text{xfList}) \otimes u\text{Input}) \langle \partial_t \eta \rangle \rrbracket_t (F r) \geq 0$

using *solves-store-ivpD(2) tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $\ast:\forall r\in\{0..t\}. \llbracket \partial_t \eta \rrbracket_t (F r) \geq 0$ **by** (*simp*)
from *obs2* **and** *tcHyp* **have** $\forall r\in\{0..t\}. ((\lambda s. \llbracket \eta \rrbracket_t (F s)) \text{ has-derivative } (\lambda x. x \ast_R (\llbracket \partial_t \eta \rrbracket_t (F r))))$ (*at r within \{0..t\}*) **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 *mvt-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 \ast *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $\llbracket \eta \rrbracket_t c \geq 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(45)
not-le)
qed
qed

lemma less-pval-to-tval:

assumes $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \upharpoonright \partial_P (\vartheta \prec \eta) \rrbracket_P st$
shows $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle_t st \geq 0$
using *assms* **by** (*auto*)

lemma leq-pval-to-tval:

assumes $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \upharpoonright \partial_P (\vartheta \preceq \eta) \rrbracket_P st$
shows $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle_t st \geq 0$
using *assms* **by** (*auto*)

lemma dInv-prelim:

assumes *substHyp*: $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0) \longrightarrow$
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \upharpoonright \partial_P \varphi \rrbracket_P st$
and *propVarsHyp*: $\text{propVars } \varphi \subseteq (\text{UNIV} - \text{varDiffs})$
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* φ)
case (*Eq* $\vartheta \eta$)
hence *hyp*: $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0) \longrightarrow$
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \upharpoonright \partial_P (\vartheta \doteq \eta) \rrbracket_P st$ **by** *blast*
then have $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0) \longrightarrow$
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList}) \otimes \text{uInput}) \langle \partial_t (\vartheta \oplus (\ominus \eta)) \rangle_t st = 0$ **by** *simp*
also have $\text{trmVars } (\vartheta \oplus (\ominus \eta)) \subseteq \text{UNIV} - \text{varDiffs}$ **using** *Eq.prem(2)* **by** *simp*
moreover have $\llbracket \vartheta \oplus (\ominus \eta) \rrbracket_t a = 0$ **using** *Eq.prem(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 $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1 \llbracket \text{set xfList} \rrbracket)) \longrightarrow st (\partial str) = 0) \longrightarrow$
 $0 \leq (\llbracket (\text{map } (\text{vdiff} \circ \pi_1) \text{ xfList} \otimes \text{uInput}) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle_t st$
using *less-pval-to-tval* **by** *metis*

also from *Less.prem*s(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.prem*s(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 $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } \text{xfList}))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $0 \leq (\llbracket (\text{map } (\text{vdiff} \circ \pi_1) \text{xfList} \otimes \text{uInput}) \langle \partial_t (\eta \oplus (\ominus \vartheta)) \rangle \rrbracket_t) st$ using *leq-pval-to-tval*
 by *metis*
 also from *Leq.prem*s(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.prem*s(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* $\varphi 1 \varphi 2$)
 then show *?case* by (*simp*)
 next
 case (*Or* $\varphi 1 \varphi 2$)
 from this show *?case* by *auto*
 qed
 qed

theorem *dInv*:
 assumes $\forall st. G st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } \text{xfList}))) \longrightarrow st (\partial str) = 0 \longrightarrow$
 $\llbracket ((\text{map } (\text{vdiff} \circ \pi_1) \text{xfList}) \otimes \text{uInput}) \upharpoonright_{\partial_P} \varphi \rrbracket_P st$
 and $\text{termVarsHyp}:\text{propVars } \varphi \subseteq (\text{UNIV} - \text{varDiffs})$
 and $\text{listsHyp}:\text{map } \pi_2 \text{xfList} = \text{map tval uInput}$
 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 $(a, b) \in \lceil P \rceil$
 from this have $a\text{Hyp}:a = b \wedge P a$ by (*metis* (*full-types*) *d-p2r rdom-p2r-contents*)
 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 $a\text{Hyp}$ have $\forall c. (a, c) \in (\text{ODEsystem } \text{xfList} \text{ with } G) \longrightarrow P c$ by *blast*
 thus $(a, b) \in \text{wp } (\text{ODEsystem } \text{xfList} \text{ with } G) \lceil P \rceil$
 using $a\text{Hyp}$ by (*simp add: boxProgrPred-chrctrzn*)

qed

```

theorem dInvFinal:
assumes  $\forall st. G\ st \longrightarrow (\forall str. str \notin (\pi_1(\text{set } xfList))) \longrightarrow st\ (\partial\ str) = 0 \longrightarrow$ 
 $\llbracket ((\text{map } (vdiff \circ \pi_1)\ xfList) \otimes uInput) \upharpoonright_{\partial_P} \varphi \rrbracket_P st$ 
and termVarsHyp:  $\text{propVars } \varphi \subseteq (UNIV - \text{varDiffs})$ 
and listsHyp:  $\text{map } \pi_2\ xfList = \text{map tval } uInput$ 
and impls:  $\lceil P \rceil \subseteq \lceil F \rceil \wedge \lceil F \rceil \subseteq \lceil Q \rceil$ 
and phi-f:  $F = \llbracket \varphi \rrbracket_P$ 
shows PRE P (ODEsystem xfList with G) POST Q
apply(rule-tac  $C = \llbracket \varphi \rrbracket_P$  in dCut)
apply(subgoal-tac  $\lceil F \rceil \subseteq wp\ (ODEsystem\ xfList\ with\ G)\ \lceil F \rceil$ , simp)
using impls and phi-f apply blast
apply(subgoal-tac PRE F (ODEsystem xfList with G) POST F, simp)
apply(rule-tac  $\varphi = \varphi$  and  $uInput = uInput$  in dInv)
prefer 5 apply(subgoal-tac PRE P (ODEsystem xfList with  $(\lambda s. G\ s \wedge F\ s)$ )
POST Q, 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

0.20.5 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  $(\lambda s. s\ ''y'' < s\ ''x'' \wedge s\ ''v'' > 0)$ 
  (ODEsystem  $[(\lambda s. s\ ''v'')]$  with  $(\lambda s. \text{True})$ )
  POST  $(\lambda s. (s\ ''y'' < s\ ''x''))$ 
apply(rule-tac  $uInput = [\lambda t\ s. s\ ''v'' \cdot t + s\ ''x'']$  in dSolve-toSolveUBC)
prefer 9 subgoal by(simp add: wp-trafo 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  $f'1 = \lambda x. s\ ''v''$  and  $g'1 = \lambda x. 0$  in derivative-intros(191))
apply(rule-tac  $f'1 = \lambda x. 0$  and  $g'1 = \lambda x. 1$  in derivative-intros(194))
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 $\text{solHyp}:\varphi_s \text{ solvesTheStoreIVP } [(x, \lambda s. s \ v), (v, \lambda s. s \ a)] \text{ withInitState } s$
and $\text{tHyp}:r \leq t \text{ and } \text{rHyp}:0 \leq r \text{ and } \text{distinct}:x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin \text{varDiffs}$
shows $\varphi_s \ r \ v = s \ a \cdot r + s \ v$
proof–
from assms **have** $1:(\lambda t. \varphi_s \ t \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a) \ \{0..t\} \text{ UNIV } \wedge \varphi_s \ 0 \ v = s \ v$
by (*simp add: solvesStoreIVP-def*)
from assms **have** $\text{obs}:\forall \ r \in \{0..t\}. \varphi_s \ r \ a = s \ a$
by(*auto simp: solvesStoreIVP-def varDiffs-def*)
have $2:(\lambda t. s \ a \cdot t + s \ v) \text{ solves-ode } (\lambda t \ r. \varphi_s \ t \ a) \ \{0..t\} \text{ UNIV}$
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 $3:\text{unique-on-bounded-closed } 0 \ \{0..t\} \ (s \ v) \ (\lambda t \ r. \varphi_s \ t \ a) \text{ UNIV (if } t = 0 \text{ then } 1 \text{ else } 1/(t+1))$
apply(*simp add: ubc-definitions del: comp-apply, rule conjI*)
using $\text{rHyp } \text{tHyp } \text{obs}$ **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* $0 \ \{0..t\} \ s \ v$
 $(\lambda t \ r. \varphi_s \ t \ a) \text{ UNIV (if } t = 0 \text{ then } 1 \text{ else } 1 / (t + 1)) \ (\lambda t. \varphi_s \ t \ v)]$)
using $\text{rHyp } \text{tHyp } 1 \ 2$ **and** 3 **by** *auto*
qed

lemma *motion-with-constant-acceleration:*
 $\text{PRE } (\lambda s. s \ \text{"y"} < s \ \text{"x"} \wedge s \ \text{"v"} \geq 0 \wedge s \ \text{"a"} > 0)$
 $(\text{ODEsystem } [(\text{"x"}, (\lambda s. s \ \text{"v"})), (\text{"v"}, (\lambda s. s \ \text{"a"}))] \text{ with } (\lambda s. \text{True}))$
 $\text{POST } (\lambda s. (s \ \text{"y"} < s \ \text{"x"}))$
apply(*rule-tac uInput*=[$\lambda t \ s. s \ \text{"a"} \cdot t^2/2 + s \ \text{"v"} \cdot t + s \ \text{"x"},$
 $\lambda t \ s. s \ \text{"a"} \cdot t + s \ \text{"v"}]$ **in** *dSolve-toSolveUBC*)
prefer 9 **subgoal** **by**(*simp add: wp-trafo 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* $\varphi_s \ \text{"x"} \ - \ - \ - \ t]$)
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*: φ_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** $1:((\lambda t. \varphi_s \ t \ v) \text{ solves-ode } (\lambda t \ r. - \varphi_s \ t \ a)) \ \{0..t\} \ \text{UNIV} \wedge \varphi_s \ 0 \ v = s \ v$

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 $2:((\lambda t. - s \ a \cdot t + s \ v) \text{ solves-ode } (\lambda t \ r. - \varphi_s \ t \ a)) \ \{0..t\} \ \text{UNIV}$

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 $3:\text{unique-on-bounded-closed } 0 \ \{0..t\} \ (s \ v) \ (\lambda t \ r. - \varphi_s \ t \ a) \ \text{UNIV} \ (\text{if } t = 0 \text{ then } 1 \text{ else } 1/(t+1))$

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* $0 \ \{0..t\} \ s \ v \ (\lambda t \ r. - \varphi_s \ t \ a) \ \text{UNIV} \ (\text{if } t = 0 \text{ then } 1 \text{ else } 1 / (t + 1)) \ (\lambda t. \varphi_s \ t \ v)]$)

using *rHyp tHyp* $1 \ 2$ **and** 3 **by** *auto*

qed

lemma *single-hop-ball*:

PRE $(\lambda \ s. 0 \leq s \ \text{"x"} \wedge s \ \text{"x"} = H \wedge s \ \text{"v"} = 0 \wedge s \ \text{"g"} > 0 \wedge 1 \geq c \wedge c \geq 0)$

$((\text{ODEsystem } [(\text{"x"}, \lambda \ s. s \ \text{"v"}), (\text{"v"}, \lambda \ s. - s \ \text{"g"})] \text{ with } (\lambda \ s. 0 \leq s \ \text{"x"}));$
 $(\text{IF } (\lambda \ s. s \ \text{"x"} = 0) \ \text{THEN } (\text{"v"} ::= (\lambda \ s. - c \cdot s \ \text{"v"})) \ \text{ELSE } (\text{"v"} ::= (\lambda \ s. s \ \text{"v"})) \ \text{FI}))$

POST $(\lambda \ s. 0 \leq s \ \text{"x"} \wedge s \ \text{"x"} \leq H)$

apply(*simp, subst dS[of* $[\lambda \ t \ s. - s \ \text{"g"} \cdot t \wedge 2/2 + s \ \text{"v"} \cdot t + s \ \text{"x"}, \lambda \ t \ s. - s \ \text{"g"} \cdot t + s \ \text{"v"}]$)

— 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  $s \ X \ t \ \tau$ 
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 ( $\lambda s. s \ "x" = 0$ )
  (ODEsystem [( $"x", (\lambda s. s \ "x" + 1)$ )] with ( $\lambda s. s \ "x" \geq 0$ ))
  POST ( $\lambda s. s \ "x" \geq 0$ )

```

using dWeakening **by** blast

lemma *system-where-the-guard-implies-the-postcondition2:*

```

  PRE ( $\lambda s. s \ "x" = 0$ )
  (ODEsystem [( $"x", (\lambda s. s \ "x" + 1)$ )] with ( $\lambda s. s \ "x" \geq 0$ ))
  POST ( $\lambda s. s \ "x" \geq 0$ )

```

```

apply(clarify, simp add: p2r-def)
apply(simp add: rel-ad-def rel-antidomain-kleene-algebra.addual.ars-r-def)
apply(simp add: rel-antidomain-kleene-algebra.fbox-def)
apply(simp add: relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def)
by auto

```

— Example of system proved with a differential invariant.

lemma *circular-motion:*

```

  PRE ( $\lambda s. (s \ "x") \cdot (s \ "x") + (s \ "y") \cdot (s \ "y") - (s \ "r") \cdot (s \ "r") = 0$ )
  (ODEsystem [( $"x", (\lambda s. s \ "y")$ ), ( $"y", (\lambda s. -s \ "x")$ )] with  $G$ )
  POST ( $\lambda s. (s \ "x") \cdot (s \ "x") + (s \ "y") \cdot (s \ "y") - (s \ "r") \cdot (s \ "r") = 0$ )

```

```

apply(rule-tac  $\eta = (t_V \ "x") \odot (t_V \ "x") \oplus (t_V \ "y") \odot (t_V \ "y") \oplus (\ominus(t_V \ "r")) \odot (t_V \ "r")$ )

```

```

  and  $uInput = [t_V \ "y", \ominus(t_V \ "x)]$  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.

declare d-p2r [simp del]

lemma *motion-with-constant-velocity-and-invariants:*

```

  PRE ( $\lambda s. s \ "x" > s \ "y" \wedge s \ "v" > 0$ )
  (ODEsystem [( $"x", \lambda s. s \ "v"$ )] with ( $\lambda s. \text{True}$ ))
  POST ( $\lambda s. s \ "x" > s \ "y"$ )

```

```

apply(rule-tac  $C = \lambda s. s \ "v" > 0$  in dCut)

```

```

apply(rule-tac  $\varphi = (t_C \ 0) \prec (t_V \ "v")$  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 $C = \lambda s. s \text{ ''}x'' > s \text{ ''}y''$ in dCut)
apply(rule-tac $\varphi = (t_V \text{ ''}y'') \prec (t_V \text{ ''}x')$ and $uInput = [t_V \text{ ''}v'']$ and
 $F = \lambda s. s \text{ ''}x'' > s \text{ ''}y''$ in dInvFinal)
apply(simp-all add: vdiff-def 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-def 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-def 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-def 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:

$PRE (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' = H \wedge s \text{ ''}v'' = 0 \wedge s \text{ ''}g'' > 0 \wedge 1 \geq c \wedge c \geq 0)$
 $((ODEsystem [(\text{''}x'', \lambda s. s \text{ ''}v'')), (\text{''}v'', \lambda s. -s \text{ ''}g'')] \text{ with } (\lambda s. 0 \leq s \text{ ''}x''));$
 $(IF (\lambda s. s \text{ ''}x'' = 0) THEN (\text{''}v'' ::= (\lambda s. -c \cdot s \text{ ''}v'')) ELSE (\text{''}v'' ::= (\lambda s. s \text{ ''}v'')) FI))$
 $POST (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H)$
apply(simp add: d-p2r, subgoal-tac $\text{rdom } [\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' = H \wedge s \text{ ''}v'' = 0 \wedge 0 < s \text{ ''}g'' \wedge c \leq 1 \wedge 0 \leq c]$
 $\subseteq wp (ODEsystem [(\text{''}x'', \lambda s. s \text{ ''}v'')), (\text{''}v'', \lambda s. -s \text{ ''}g'')] \text{ with } (\lambda s. 0 \leq s \text{ ''}x''))$
 $([inf (sup (- (\lambda s. s \text{ ''}x'' = 0)) (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H)) (sup (\lambda s. s \text{ ''}x'' = 0) (\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H))])$
apply(simp add: d-p2r, 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-def 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-def 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-def 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*: $0 \leq x \implies 0 < g \implies 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \implies (x::\text{real}) \leq H$

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** $(v \cdot v)/(2 \cdot g) \geq 0$

by (*meson divide-nonneg-pos linordered-field-class.sign-simps(44) zero-less-numeral*)

ultimately have $H - x \geq 0$ **by** *linarith*

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

qed

lemma *bouncing-ball*:

PRE $(\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' = H \wedge s \text{ ''}v'' = 0 \wedge s \text{ ''}g'' > 0)$

$((ODEsystem [(\text{''}x'', \lambda s. s \text{ ''}v''), (\text{''}v'', \lambda s. - s \text{ ''}g'')]$ *with* $(\lambda s. 0 \leq s \text{ ''}x'')$);

$(IF (\lambda s. s \text{ ''}x'' = 0) THEN (\text{''}v'' ::= (\lambda s. - s \text{ ''}v'')) ELSE (Id) FI))^*$

POST $(\lambda s. 0 \leq s \text{ ''}x'' \wedge s \text{ ''}x'' \leq H)$

apply(*rule rel-antidomain-kleene-algebra.fbox-starI[of - $\lceil \lambda s. 0 \leq s \text{ ''}x'' \wedge 0 < s \text{ ''}g'' \wedge$*

$2 \cdot s \text{ ''}g'' \cdot s \text{ ''}x'' = 2 \cdot s \text{ ''}g'' \cdot H - (s \text{ ''}v'' \cdot s \text{ ''}v'')]$)

apply(*simp, simp add: d-p2r*)

apply(*subgoal-tac*

rdm $\lceil \lambda s. 0 \leq s \text{ ''}x'' \wedge 0 < s \text{ ''}g'' \wedge 2 \cdot s \text{ ''}g'' \cdot s \text{ ''}x'' = 2 \cdot s \text{ ''}g'' \cdot H - s \text{ ''}v'' \cdot s \text{ ''}v'' \rceil$

$\subseteq wp (ODEsystem [(\text{''}x'', \lambda s. s \text{ ''}v''), (\text{''}v'', \lambda s. - s \text{ ''}g'')]$ *with* $(\lambda s. 0 \leq s \text{ ''}x'')$

$)$
 $\lceil inf (sup (- (\lambda s. s \text{ ''}x'' = 0)) (\lambda s. 0 \leq s \text{ ''}x'' \wedge 0 < s \text{ ''}g'' \wedge 2 \cdot s \text{ ''}g'' \cdot s \text{ ''}x'' =$

$2 \cdot s \text{ ''}g'' \cdot H - s \text{ ''}v'' \cdot s \text{ ''}v''))$

$(sup (\lambda s. s \text{ ''}x'' = 0) (\lambda s. 0 \leq s \text{ ''}x'' \wedge 0 < s \text{ ''}g'' \wedge 2 \cdot s \text{ ''}g'' \cdot s \text{ ''}x'' =$

$2 \cdot s \text{ ''}g'' \cdot H - s \text{ ''}v'' \cdot s \text{ ''}v''))]$

apply(*simp add: d-p2r*)

apply(*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-def varDiffs-def, clarify, erule-tac $x=\text{''}g''$ in allE, simp*)

apply(*rule-tac $C = \lambda s. 2 \cdot s \text{ ''}g'' \cdot s \text{ ''}x'' = 2 \cdot s \text{ ''}g'' \cdot H - s \text{ ''}v'' \cdot s \text{ ''}v''$ in dCut*)

apply(*rule-tac $\varphi = (t_C 2) \odot (t_V \text{ ''}g'') \odot (t_C H) \oplus (\ominus ((t_V \text{ ''}v'') \odot (t_V \text{ ''}v'')))$*

$\doteq (t_C 2) \odot (t_V \text{ ''}g'') \odot (t_V \text{ ''}x'')$ **and** $uInput=[t_V \text{ ''}v'', \ominus t_V \text{ ''}g'']$ *in dInvFinal*)

apply(*simp-all add: vdiff-def varDiffs-def, clarify, erule-tac $x=\text{''}g''$ in allE, simp*)


```
apply(rule dWeakening, clarsimp)  
using bouncing-ball-invariant by auto  
  
declare d-p2r [simp]  
  
end
```