

(Extended) Interval Analysis

Achim D. Brucker^{ORCID}

Amy Stell^{ORCID}

April 18, 2024

Department of Computer Science
University of Exeter
Exeter, UK
{a.brucker,a.stell}@exeter.ac.uk

Abstract

Interval analysis (also called interval arithmetic) is a well known mathematical technique to analyse or mitigate rounding errors or measurement errors. Thus, it is promising to integrate interval analysis into program verification environments. Such an integration is not only useful for the verification of numerical algorithms: the need to ensure that computations stay within certain bounds is common. For example to show that computations stay within the hardware bounds of a given number representation.

Another application is the verification of cyber-physical systems, where a discretised implementation approximates a system described in physical quantities expressed using perfect mathematical reals, and perfect ordinary differential equations.

In this AFP entry, we formalise extended interval analysis, including the concept of inclusion isotone (or inclusion isotonic) (extended) interval analysis. The main result is the formal proof that interval-splitting converges for Lipschitz-continuous interval isotone functions. From pragmatic perspective, we provide the datatypes and theory required for integrating interval analysis into other formalisations and applications.

Keywords: Extended Interval Analysis, Formalising Mathematics, Isabelle/HOL

Contents

1	Introduction	9
2	Interval Utilities (Interval_Utilities)	13
2.1	Preliminaries	13
2.2	Interval Bounds and Set Conversion	14
2.3	Linear Order on List of Intervals	15
2.4	Support for Lists of Intervals	17
2.5	Interval Width and Arithmetic Operations	17
2.6	Interval Multiplication	18
2.7	Distance-based Properties of Intervals	18
3	Basic Properties of Interval Division (Interval_Division_Non_Zero)	19
3.1	Preliminaries	19
3.2	A Locale for Interval Division Where the Quotient-Interval does not Contain Zero	19
4	A Naive Interval Division for Real Intervals (Interval_Division_Real)	23
5	Affine Functions (Affine_Functions)	25
5.1	Definition of Affine Functions, Alternative Definitions, and Special Cases	25
5.2	Common Linear Polynomial Functions	25
5.3	Linear Polynomial Functions and Orderings	26
6	Interval Inclusion Isotonicity (Inclusion_Isotonicity)	29
6.1	Interval Extension	29
6.1.1	Textbook Definition of Interval Extension	29
6.1.2	A Stronger Definition of Interval Extension	29
6.2	Interval Inclusion Isotonicity	31
6.2.1	Compositionality of Interval Inclusion Isotonicity	33
6.2.2	Interval Inclusion Isotonicity of the Core Operator	33
6.2.3	Interval Inclusion Isotonicity of Various Functions	34
6.3	Interval Extension and Inclusion Properties	36
6.4	Division	38
7	Lipschitz Continuity of Intervals (Lipschitz_Interval_Extension)	39
7.1	Definition of Lipschitz Continuity on Intervals	39
7.1.1	Lipschitz Continuity of Operations	40
7.1.2	Interval bounds on reals	41
8	Multi-Intervals (Multi_Interval_Preliminaries)	43
8.1	Preliminaries	43
8.1.1	A Class for Capturing Monotonicity of Minus	43
8.1.2	Infrastructure for Lifting Interval Operations to Lists of Intervals	43
8.1.3	Utilities for (Sorted) Lists of Intervals	45

8.1.4	Various Notions of Validity of Sorted Lists of Intervals	50
8.1.5	Union over a List of Intervals	54
9	Subdivisions and Refinements (☞Lipschitz_Subdivisions_Refinements)	59
9.1	Subdivisions	59
9.2	Refinement	61
9.3	Lipschitz Interval Inclusive	68
9.4	Lipschitz Convergence	71
10	Interval Analysis (☞Interval_Analysis)	73
11	Extended Division on Intervals (☞Extended_Interval_Division)	75
12	Extended Interval Analysis (☞Extended_Interval_Analysis)	77
12.1	Overlapping Multi-Intervals (☞Multi_Interval_Overlapping)	77
12.1.1	Type Definition	77
12.1.2	Equality and Orderings	77
12.1.3	Zero and One	79
12.1.4	Addition	80
12.1.5	Unary Minus	80
12.1.6	Subtraction	81
12.1.7	Multiplication	81
12.1.8	Multiplicative Inverse and Division	81
12.1.9	Membership	82
12.2	Non-Overlapping Multi-Intervals (☞Multi_Interval_Non_Overlapping)	82
12.2.1	Type Definition	82
12.2.2	Equality and Orderings	82
12.2.3	Zero and One	85
12.2.4	Addition	85
12.2.5	Unary Minus	86
12.2.6	Subtraction	86
12.2.7	Multiplication	86
12.2.8	Multiplicative Inverse and Division	87
12.2.9	Membership	87
12.3	Adjacent Multi-Intervals (☞Multi_Interval_Adjacent)	87
12.3.1	A Type For Non Overlapping Multi Intervals	88
12.3.2	Equality and Orderings	88
12.3.3	Zero and One	90
12.3.4	Addition	91
12.3.5	Unary Minus	91
12.3.6	Subtraction	91
12.3.7	Multiplication	92
12.3.8	Multiplicative Inverse and Division	92
12.3.9	Membership	92
12.4	Bringing Everything Together (☞Multi_Interval)	93
13	Extended Division on Multi-Intervals (☞Extended_Multi_Interval_Division_Core)	95
13.1	Division over List of Intervals	95

13.2	Multi-Interval Division	95
13.2.1	Overlapping Multi-Intervals (📄Extended_Multi_Interval_Division_Overlapping)	95
13.2.2	Non Overlapping Multi-Intervals (📄Extended_Multi_Interval_Division_Non_Overlapping)	96
13.2.3	Adjacent Multi-Intervals (📄Extended_Multi_Interval_Division_Adjacent)	96
13.3	Bringing Everything Together (📄Extended_Multi_Interval_Division)	96
14	Extended Multi-Interval Analysis (📄Extended_Multi_Interval_Analysis)	99

1 Introduction

Interval analysis [4] in general and, in particular, inclusion isotone extended interval arithmetic [5] are well known mathematical techniques to analyse or mitigate rounding errors or measurement errors. Thus, it is promising to integrate interval analysis into program verification environments. Such an integration is not only useful for the verification of numerical algorithms: the need to ensure that computations stay within certain bounds is common. For example to show that computations stay within the hardware bounds of a given number representation.

Another application is the verification of cyber-physical systems, where a discretised implementation approximates a system described in physical quantities expressed using perfect mathematical reals, and perfect ordinary differential equations. Moreover, first applications of interval analysis to the security analysis of neural networks are promising [6, 3] and, when combined with existing works of verifying neural networks in Isabelle [2] provide a verification approach using a “correct-by-construction” verification tool for neural networks.

In this AFP entry, we formalise extended interval analysis, including the concept of inclusion isotone (extended) interval analysis. The main result is the formal proof that interval-splitting converges for Lipschitz-continuous inclusion isotone functions. From pragmatic perspective, we provide the datatypes and theory required for integrating interval analysis into other formalisations and applications. In more detail, our contributions are:

1. a conservative formalisation of (extended) interval arithmetic in Isabelle/HOL, including inclusion isotonicity;
2. we formally prove that interval-splitting converges for Lipschitz-continuous inclusion isotone functions;

From an end-user’s perspective, the main entry points into this session are the following three theories:

- `Interval_Analysis` (Chapter 10): This theory provides interval analysis over standard types such as real or integer. All operations work over (closed) intervals.
- `Extended_Interval_Analysis` (Chapter 12): This theory provides extended interval analysis over the type extended reals. All operations work over (closed) intervals.
- `Extended_Multi_Interval_Analysis` (Chapter 14): This theory provides extended multi-interval analysis over the type extended reals. All operations work over multi-intervals, i.e., lists of (closed) intervals.

The following publication [1] gives a high-level overview of this AFP entry:

A. D. Brucker, T. Cameron-Burke, and A. Stell. Formally verified interval arithmetic and its application to program verification. In 13th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE 2024). IEEE, 2024.

The rest of this document is automatically generated from the formalisation in Isabelle/HOL, i.e., all content is checked by Isabelle. Overall, the structure of this document follows the theory dependencies (see Figure 1.1).

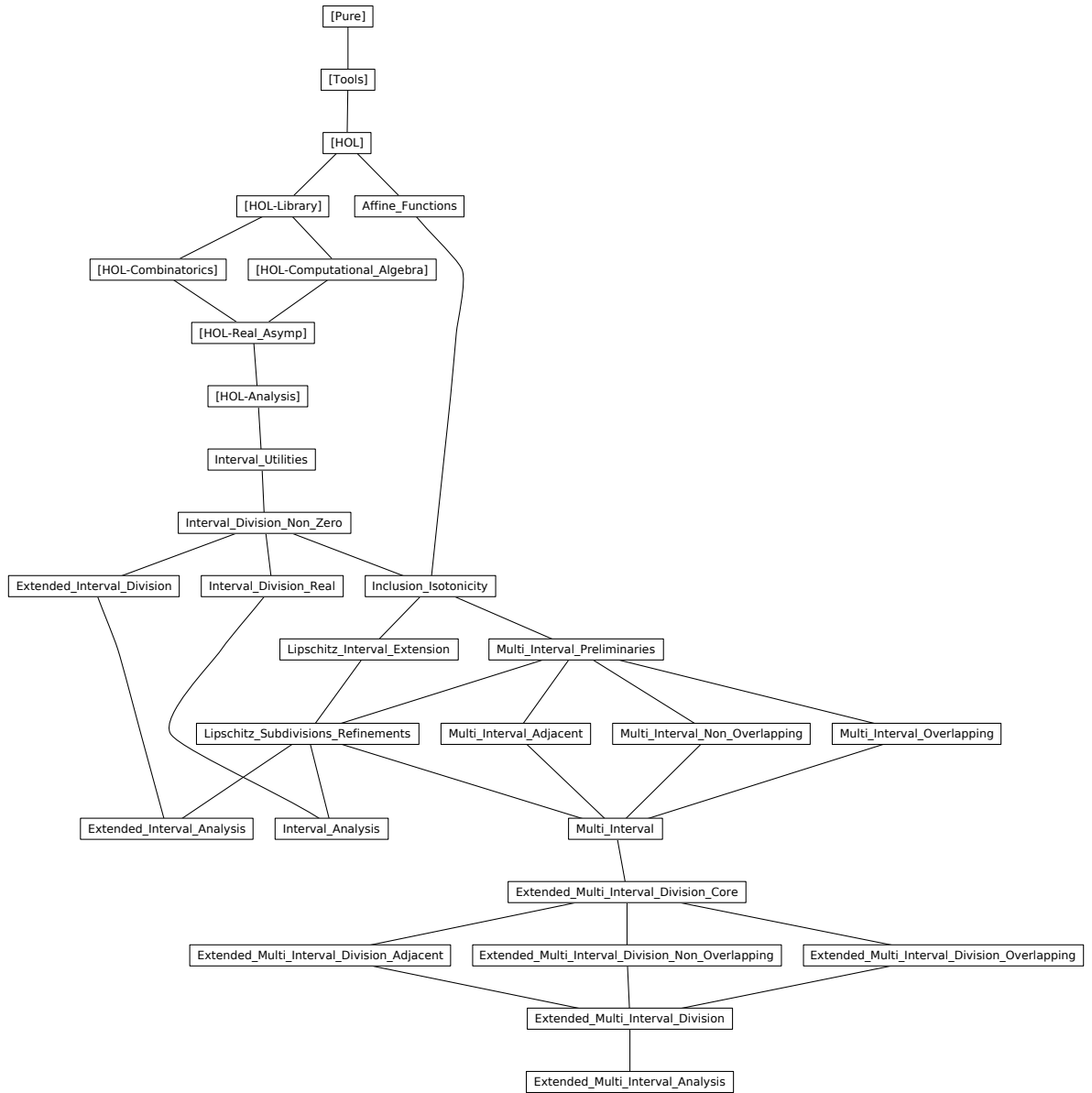


Figure 1.1: The Dependency Graph of the Isabelle Theories.

Generated Sessions

2 Interval Utilities (Interval_Utilities)

theory

Interval_Utilities

imports

HOL-Library.Interval

HOL-Analysis.Analysis

HOL-Library.Interval_Float

begin

2.1 Preliminaries

lemma *compact_set_of*:

fixes $X :: \langle 'a :: \{preorder, topological_space, ordered_euclidean_space\} \text{ interval} \rangle$

shows $\langle compact \ (set_of\ X) \rangle$

$\langle proof \rangle$

lemma *bounded_set_of*:

fixes $X :: \langle 'a :: \{preorder, topological_space, ordered_euclidean_space\} \text{ interval} \rangle$

shows $\langle bounded \ (set_of\ X) \rangle$

$\langle proof \rangle$

lemma *compact_img_set_of*:

fixes $X :: \langle real \ interval \rangle$ and $f :: \langle real \Rightarrow real \rangle$

assumes $\langle continuous_on \ (set_of\ X) \ f \rangle$

shows $\langle compact \ (f' \ set_of\ X) \rangle$

$\langle proof \rangle$

lemma *sup_inf_dist_bounded*:

fixes $X :: \langle real \ set \rangle$

shows $\langle bdd_below\ X \implies bdd_above\ X \implies \forall x \in X. \forall x' \in X. dist\ x\ x' \leq Sup\ X - Inf\ X \rangle$

$\langle proof \rangle$

lemma *set_of_nonempty[simp]*: $\langle set_of\ X \neq \{\} \rangle$

$\langle proof \rangle$

lemma *lower_in_interval[simp]*: $\langle lower\ X \in_i\ X \rangle$

$\langle proof \rangle$

lemma *upper_in_interval[simp]*: $\langle upper\ X \in_i\ X \rangle$

$\langle proof \rangle$

lemma *bdd_below_set_of*: $\langle bdd_below \ (set_of\ X) \rangle$

$\langle proof \rangle$

lemma *bdd_above_set_of*: $\langle bdd_above \ (set_of\ X) \rangle$

$\langle proof \rangle$

lemma *closed_set_of*: $\langle \text{closed } (\text{set_of } (X::\text{real interval})) \rangle$
<proof>

lemma *set_f_nonempty*: $\langle f \text{ ' set_of } X \neq \{\} \rangle$
<proof>

lemma *interval_linorder_case_split*[*case_names LeftOf Including RightOf*]:
assumes $\langle \text{upper } x < c \implies P (x::('a::\text{linorder interval})) \rangle$
 $\langle (c \in_i x \implies P x) \rangle$
 $\langle (c < \text{lower } x \implies P x) \rangle$
shows $\langle P x \rangle$
<proof>

lemma *foldl_conj_True*:
 $\langle \text{foldl } (\wedge) x xs = \text{list_all } (\lambda e. e = \text{True}) (x\#xs) \rangle$
<proof>

lemma *foldl_conj_set_True*:
 $\langle \text{foldl } (\wedge) x xs = (\forall e \in \text{set } (x\#xs). e = \text{True}) \rangle$
<proof>

2.2 Interval Bounds and Set Conversion

lemma *sup_set_of*:
fixes $X :: 'a::\{\text{conditionally_complete_lattice}\} \text{ interval}$
shows $\text{Sup } (\text{set_of } X) = \text{upper } X$
<proof>

lemma *inf_set_of*:
fixes $X :: 'a::\{\text{conditionally_complete_lattice}\} \text{ interval}$
shows $\text{Inf } (\text{set_of } X) = \text{lower } X$
<proof>

lemma *inf_le_sup_set_of*:
fixes $X :: 'a::\{\text{conditionally_complete_lattice}\} \text{ interval}$
shows $\text{Inf } (\text{set_of } X) \leq \text{Sup } (\text{set_of } X)$
<proof>

lemma *in_bounds*: $\langle x \in_i X \implies \text{lower } X \leq x \wedge x \leq \text{upper } X \rangle$
<proof>

lemma *lower_bounds*[*simp*]:
assumes $\langle L \leq U \rangle$
shows $\langle \text{lower } (\text{Interval}(L,U)) = L \rangle$
<proof>

lemma *upper_bounds* [*simp*]:
assumes $\langle L \leq U \rangle$
shows $\langle \text{upper } (\text{Interval}(L,U)) = U \rangle$
<proof>

lemma *lower_point_interval*[*simp*]: $\langle \text{lower } (\text{Interval } (x,x)) = x \rangle$
<proof>

lemma *upper_point_interval*[simp]: $\langle \text{upper } (\text{Interval } (x,x)) = x \rangle$
<proof>

lemma *map2_nth*:

assumes $\langle \text{length } xs = \text{length } ys \rangle$
and $\langle n < \text{length } xs \rangle$
shows $\langle (\text{map2 } f \text{ } xs \text{ } ys)!n = f (xs!n) (ys!n) \rangle$
<proof>

lemma *map_set*: $\langle a \in \text{set } (\text{map } f \text{ } X) \implies (\exists x \in \text{set } X . f \text{ } x = a) \rangle$
<proof>

lemma *map_pair_set_left*: $\langle (a,b) \in \text{set } (\text{zip } (\text{map } f \text{ } X) (\text{map } f \text{ } Y)) \implies (\exists x \in \text{set } X . f \text{ } x = a) \rangle$
<proof>

lemma *map_pair_set_right*: $\langle (a,b) \in \text{set } (\text{zip } (\text{map } f \text{ } X) (\text{map } f \text{ } Y)) \implies (\exists y \in \text{set } Y . f \text{ } y = b) \rangle$
<proof>

lemma *map_pair_set*: $\langle (a,b) \in \text{set } (\text{zip } (\text{map } f \text{ } X) (\text{map } f \text{ } Y)) \implies (\exists x \in \text{set } X . f \text{ } x = a) \wedge (\exists y \in \text{set } Y . f \text{ } y = b) \rangle$
<proof>

lemma *map_pair_f_all*:

assumes $\langle \text{length } X = \text{length } Y \rangle$
shows $\langle (\forall (x,y) \in \text{set } (\text{zip } (\text{map } f \text{ } X) (\text{map } f \text{ } Y)) . x \leq y) = (\forall (x,y) \in \text{set } (\text{zip } X \text{ } Y) . f \text{ } x \leq f \text{ } y) \rangle$
<proof>

definition *map_interval_swap* :: $\langle 'a::\text{linorder} \times 'a \rangle \text{ list} \Rightarrow 'a \text{ interval list} \rangle$ **where**
 $\langle \text{map_interval_swap} = \text{map } (\lambda (x,y) . \text{Interval } (\text{if } x \leq y \text{ then } (x,y) \text{ else } (y,x))) \rangle$

definition *mk_interval* :: $\langle 'a::\text{linorder} \times 'a \rangle \Rightarrow 'a \text{ interval} \rangle$ **where**
 $\langle \text{mk_interval} = (\lambda (x,y) . \text{Interval } (\text{if } x \leq y \text{ then } (x,y) \text{ else } (y,x))) \rangle$

definition *mk_interval'* :: $\langle 'a::\text{linorder} \times 'a \rangle \Rightarrow 'a \text{ interval} \rangle$ **where**
 $\langle \text{mk_interval}' = (\lambda (x,y) . (\text{if } x \leq y \text{ then } \text{Interval}(x,y) \text{ else } \text{Interval}(y,x))) \rangle$

lemma *map_interval_swap_code*[code]:

$\langle \text{map_interval_swap} = \text{map } (\lambda (x,y) . \text{the } (\text{if } x \leq y \text{ then } \text{Interval}' \text{ } x \text{ } y \text{ else } \text{Interval}' \text{ } y \text{ } x)) \rangle$
<proof>

lemma *mk_interval_code*[code]:

$\langle \text{mk_interval} = (\lambda (x,y) . \text{the } (\text{if } x \leq y \text{ then } \text{Interval}' \text{ } x \text{ } y \text{ else } \text{Interval}' \text{ } y \text{ } x)) \rangle$
<proof>

lemma *mk_interval'*:

$\langle \text{mk_interval} = (\lambda (x,y) . (\text{if } x \leq y \text{ then } \text{Interval}(x,y) \text{ else } \text{Interval}(y,x))) \rangle$
<proof>

lemma *mk_interval_lower*[simp]: $\langle \text{lower } (\text{mk_interval } (x,y)) = (\text{if } x \leq y \text{ then } x \text{ else } y) \rangle$
<proof>

lemma *mk_interval_upper*[simp]: $\langle \text{upper } (\text{mk_interval } (x,y)) = (\text{if } x \leq y \text{ then } y \text{ else } x) \rangle$
<proof>

2.3 Linear Order on List of Intervals

definition

le_interval_list :: $\langle 'a::\text{linorder} \rangle \text{ interval list} \Rightarrow 'a \text{ interval list} \Rightarrow \text{bool}$ $((_ / \leq_I _)$ [51, 51] 50)

where

$\langle \text{le_interval_list } Xs \ Ys \equiv (\text{length } Xs = \text{length } Ys) \wedge (\text{foldl } (\wedge) \ \text{True } (\text{map2 } (\leq) \ Xs \ Ys)) \rangle$

lemma le_interval_single: $\langle (x \leq y) = ([x] \leq_I [y]) \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_empty[simp]: $\langle [] \leq_I [] \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_rev: $\langle (is \leq_I js) = (\text{rev } is \leq_I \text{rev } js) \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_imp_length:

assumes $\langle Xs \leq_I Ys \rangle$ **shows** $\langle \text{length } Xs = \text{length } Ys \rangle$

$\langle \text{proof} \rangle$

lemma lsplit_left: **assumes** $\langle \text{length } (xs) = \text{length } (ys) \rangle$

and $\langle (\forall n < \text{length } (x \# xs). (x \# xs) ! n \leq (y \# ys) ! n) \rangle$ **shows** \langle

$((\forall n < \text{length } xs. xs!n \leq ys!n) \wedge x \leq y) \rangle$

$\langle \text{proof} \rangle$

lemma lsplit_right: **assumes** $\langle \text{length } (xs) = \text{length } (ys) \rangle$

and $\langle (\forall n < \text{length } xs. xs!n \leq ys!n) \wedge x \leq y \rangle$

shows $\langle (n < \text{length } (x \# xs) \longrightarrow (x \# xs) ! n \leq (y \# ys) ! n) \rangle$

$\langle \text{proof} \rangle$

lemma lsplit: **assumes** $\langle \text{length } (xs) = \text{length } (ys) \rangle$

shows $\langle (\forall n < \text{length } (x \# xs). (x \# xs) ! n \leq (y \# ys) ! n) =$

$((\forall n < \text{length } xs. xs!n \leq ys!n) \wedge x \leq y) \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_all':

assumes $\langle \text{length } Xs = \text{length } Ys \rangle$ **and** $\langle Xs \leq_I Ys \rangle$ **shows** $\langle \forall n < \text{length } Xs. Xs!n \leq Ys!n \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_all2:

assumes $\langle \text{length } Xs = \text{length } Ys \rangle$

and $\langle \forall n < \text{length } Xs. (Xs!n \leq Ys!n) \rangle$

shows $\langle Xs \leq_I Ys \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_all:

assumes $\langle Xs \leq_I Ys \rangle$ **shows** $\langle \forall n < \text{length } Xs. Xs!n \leq Ys!n \rangle$

$\langle \text{proof} \rangle$

lemma le_interval_list_imp:

assumes $\langle Xs \leq_I Ys \rangle$ **shows** $\langle n < \text{length } Xs \longrightarrow Xs!n \leq Ys!n \rangle$

$\langle \text{proof} \rangle$

lemma interval_set_leq_eq: $\langle (X \leq Y) = (\text{set_of } X \subseteq \text{set_of } Y) \rangle$

for $X :: 'a::\text{linordered_ring interval}$

$\langle \text{proof} \rangle$

lemma *times_interval_right*:
fixes $X Y C :: \langle 'a :: \text{linordered_ring interval} \rangle$
assumes $\langle X \leq Y \rangle$
shows $\langle C * X \leq C * Y \rangle$
 $\langle \text{proof} \rangle$

lemma *times_interval_left*:
fixes $X Y C :: \langle 'a :: \{\text{real_normed_algebra, linordered_ring, linear_continuum_topology}\} \text{interval} \rangle$
assumes $\langle X \leq Y \rangle$
shows $\langle X * C \leq Y * C \rangle$
 $\langle \text{proof} \rangle$

2.4 Support for Lists of Intervals

abbreviation *in_interval_list*:: $\langle ('a :: \text{preorder}) \text{list} \Rightarrow 'a \text{ interval list} \Rightarrow \text{bool} \rangle ((_ / \in_I _) [51, 51] 50)$
where $\langle \text{in_interval_list } xs \ Xs \equiv \text{foldl } (\wedge) \ \text{True } (\text{map2 } (\text{in_interval}) \ xs \ Xs) \rangle$

lemma *interval_of_in_interval_list*[simp]: $\langle xs \in_I \ \text{map } \text{interval_of } \ xs \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_of_in_eq*: $\langle \text{interval_of } x \leq X = (x \in_i X) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_of_list_in*:
assumes $\langle \text{length } \text{inputs} = \text{length } \text{Inputs} \rangle$
shows $\langle (\text{map } \text{interval_of } \ \text{inputs} \leq_I \ \text{Inputs}) = (\text{inputs} \in_I \ \text{Inputs}) \rangle$
 $\langle \text{proof} \rangle$

2.5 Interval Width and Arithmetic Operations

lemma *interval_width_addition*:
fixes $A :: 'a :: \{\text{linordered_ring}\} \text{interval}$
shows $\langle \text{width } (A + B) = \text{width } A + \text{width } B \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_width_times*:
fixes $a :: 'a :: \{\text{linordered_ring}\}$ **and** $A :: 'a \text{ interval}$
shows $\langle \text{width } (\text{interval_of } a * A) = |a| * \text{width } A \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_sup_width*:
fixes $X Y :: 'a :: \{\text{linordered_ring, lattice}\} \text{interval}$
shows $\langle \text{width } (\text{sup } X Y) = \max (\text{upper } X) (\text{upper } Y) - \min (\text{lower } X) (\text{lower } Y) \rangle$
 $\langle \text{proof} \rangle$

lemma *width_expanded*: $\langle \text{interval_of } (\text{width } Y) = \text{Interval}(\text{upper } Y - \text{lower } Y, \text{upper } Y - \text{lower } Y) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_width_positive*:
fixes $X :: \langle 'a :: \{\text{linordered_ring}\} \text{interval} \rangle$
shows $\langle 0 \leq \text{width } X \rangle$

⟨proof⟩

2.6 Interval Multiplication

lemma *interval_interval_times*:

⟨ $X * Y = \text{Interval}(\text{Min} \{(\text{lower } X * \text{lower } Y), (\text{lower } X * \text{upper } Y), (\text{upper } X * \text{lower } Y), (\text{upper } X * \text{upper } Y)\},$
 $\text{Max} \{(\text{lower } X * \text{lower } Y), (\text{lower } X * \text{upper } Y), (\text{upper } X * \text{lower } Y), (\text{upper } X * \text{upper } Y)\})$ ⟩

⟨proof⟩

lemma *interval_times_scalar*: ⟨ $\text{interval_of } a * A = \text{mk_interval}(a * \text{lower } A, a * \text{upper } A)$ ⟩

⟨proof⟩

lemma *interval_times_scalar_pos_l*:

fixes $a :: 'a::\{\text{ordered_semiring}, \text{times}, \text{linorder}\}$

assumes $\langle 0 \leq a \rangle$

shows ⟨ $\text{interval_of } a * A = \text{Interval}(a * \text{lower } A, a * \text{upper } A)$ ⟩

⟨proof⟩

lemma *interval_times_scalar_pos_r*:

fixes $a :: 'a::\{\text{linordered_idom}\}$

assumes $\langle 0 \leq a \rangle$

shows ⟨ $A * \text{interval_of } a = \text{Interval}(a * \text{lower } A, a * \text{upper } A)$ ⟩

⟨proof⟩

2.7 Distance-based Properties of Intervals

Given two real intervals X and Y , and two real numbers a and b , the width of the sum of the scaled intervals is equivalent to the width of the two individual intervals.

lemma *width_of_scaled_interval_sum*:

fixes $X :: 'a::\{\text{linordered_ring}\}$ *interval*

shows ⟨ $\text{width}(\text{interval_of } a * X + \text{interval_of } b * Y) = |a| * \text{width } X + |b| * \text{width } Y$ ⟩

⟨proof⟩

lemma *width_of_product_interval_bound_real*:

fixes $X :: \text{real interval}$

shows ⟨ $\text{interval_of}(\text{width}(X * Y)) \leq \text{abs_interval}(X) * \text{interval_of}(\text{width } Y) + \text{abs_interval}(Y) * \text{interval_of}(\text{width } X)$ ⟩

⟨proof⟩

lemma *width_of_product_interval_bound_int*:

fixes $X :: \text{int interval}$

shows ⟨ $\text{interval_of}(\text{width}(X * Y)) \leq \text{abs_interval}(X) * \text{interval_of}(\text{width } Y) + \text{abs_interval}(Y) * \text{interval_of}(\text{width } X)$ ⟩

⟨proof⟩

end

3 Basic Properties of Interval Division

(Interval_Division_Non_Zero)

theory

Interval_Division_Non_Zero

imports

Interval_Utilities

begin

The theory *HOL—Library.Interval* does not define a division operation on intervals. In the following we define a locale capturing the core properties of division by an interval that does not contain zero.

3.1 Preliminaries

lemma *division_leq_neg*:

fixes $x :: 'a::\{\text{linordered_field}\}$

assumes $0 < x$ and $y < 0$ and $z < 0$ and $y \leq z$

shows $x / z \leq x / y$

<proof>

lemma *division_leq*:

fixes $x :: 'a::\{\text{linordered_field}\}$

assumes $0 < x$ and $y \leq z$ and $\langle y \neq 0 \wedge z \neq 0 \rangle$ and $\langle (y < 0 \wedge z < 0) \vee (0 < y \wedge 0 < z) \rangle$

shows $x / z \leq x / y$

<proof>

lemma *upper_leq_lower_div*:

fixes $Y :: 'a::\{\text{linordered_field}\}$ interval

assumes $\langle \text{lower } Y \leq \text{upper } Y \rangle$ and $\langle \neg 0 \in_i Y \rangle$

shows $\langle 1 / \text{upper } Y \leq 1 / \text{lower } Y \rangle$

<proof>

3.2 A Locale for Interval Division Where the Quotient-Interval does not Contain Zero

locale *interval_division* = *inverse* +

constrains *inverse* :: $\langle 'a::\{\text{linordered_field}, \text{real_normed_algebra}, \text{linear_continuum_topology}\} \text{ interval} \Rightarrow 'a \text{ interval} \rangle$

and *divide* :: $\langle 'a::\{\text{linordered_field}, \text{real_normed_algebra}, \text{linear_continuum_topology}\} \text{ interval} \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ interval} \rangle$

assumes *inverse_left*: $\langle \neg 0 \in_i x \implies 1 \leq (\text{inverse } x) * x \rangle$

and *divide*: $\langle \neg 0 \in_i y \implies x \leq (\text{divide } x y) * y \rangle$

begin

end

lemma *interval_non_zero_eq*:

$\langle \neg 0 \in_i (i::'a::\{\text{linorder}, \text{zero}\} \text{ interval}) = (\text{lower } i < 0 \wedge \text{upper } i < 0) \vee (\text{lower } i > 0 \wedge \text{upper } i > 0) \rangle$

<proof>

lemma *inverse_includes_one*:

assumes $\neg 0 \in_i (i::'a::\{\text{division_ring}, \text{linordered_ring}\} \text{interval})$

shows $\neg 1 \in_i (\text{mk_interval } (1 / \text{upper } i), 1 / \text{lower } i) * i$

<proof>

lemma *inverse_includes_one'*:

assumes $\neg 0 \in_i (i::'a::\{\text{division_ring}, \text{linordered_ring}\} \text{interval})$

shows $\neg 1 \leq (\text{mk_interval } (1 / \text{upper } i), 1 / \text{lower } i) * i$

<proof>

locale *interval_division_inverse* = *inverse* +

constrains *inverse* :: $\langle 'a::\{\text{linordered_field}, \text{real_normed_algebra}, \text{linear_continuum_topology}\} \text{interval} \Rightarrow 'a \text{ interval} \rangle$

and *divide* :: $\langle 'a::\{\text{linordered_field}, \text{real_normed_algebra}, \text{linear_continuum_topology}\} \text{interval} \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ interval} \rangle$

assumes *inverse_non_zero_def*: $\neg 0 \in_i x \implies (\text{inverse } x) = \text{mk_interval}(1 / (\text{upper } x), 1 / (\text{lower } x))$

and *divide_non_zero_def*: $\neg 0 \in_i y \implies (\text{divide } x y) = \text{inverse } y * x$

begin

sublocale *interval_division* *divide* *inverse*

<proof>

lemma *inverse_left_ge_one*:

assumes $\neg 0 \in_i x$

shows $\neg 1 \leq (\text{inverse } x) * x$

<proof>

lemma *division_right_ge_refl*:

assumes $\neg 0 \in_i y$

shows $x \leq x * ((\text{inverse } y) * y)$

<proof>

lemma *division_right_ge_refl'*:

assumes $\neg 0 \in_i y$

shows $x \leq x * \text{inverse } y * y$

<proof>

lemma *interval_div_constant*:

assumes $\langle 0 \notin \text{set_of } Y \rangle$ **and** $\langle 0 \leq x \rangle$

shows $\langle \text{divide } (\text{interval_of } x) Y = \text{Interval}(x / \text{upper } Y, x / \text{lower } Y) \rangle$

<proof>

lemma *interval_of_width*:

assumes $\neg 0 \in_i Y$

shows $\langle \text{interval_of}(\text{width } (\text{divide } (\text{interval_of } 1) Y)) = \text{Interval}(1 / \text{lower } Y - 1 / \text{upper } Y, 1 / \text{lower } Y - 1 / \text{upper } Y) \rangle$

<proof>

lemma *abs_pos*:

assumes $\langle 0 < \text{lower } Y \rangle$ **and** $\neg 0 \in_i Y$

shows $\langle \text{abs_interval}(\text{divide } (\text{interval_of } 1) Y) = \text{Interval}(1 / \text{upper } Y, 1 / \text{lower } Y) \rangle$

<proof>

lemma *abs_neg*:

assumes $\langle \text{upper } Y < 0 \rangle$ **and** $\langle \neg 0 \in_i Y \rangle$

shows $\langle \text{abs_interval}(\text{divide}(\text{interval_of } 1) Y) = \text{Interval}(1 / |\text{lower } Y|, 1 / |\text{upper } Y|) \rangle$

$\langle \text{proof} \rangle$

end

end

4 A Naive Interval Division for Real Intervals

(Interval_Division_Real)

theory

Interval_Division_Real

imports

Interval_Division_Non_Zero

begin

The theory *HOL—Library.Interval* does not define a division operation on intervals. Actually, In the following we define division in a straight forward way. This is possible, as in HOL, the property $?a / (o::?'a) = (o::?'a)$ holds. Therefore, we do not need to use, in the first instance, extended interval analysis (e.g., based on the type *ereal*. As a consequence, results obtained using this definition might differ from results obtained using definitions of divisions using extended reals (e.g., [4]).

instantiation *interval* :: (*linordered_field*, *real_normed_algebra*, *linear_continuum_topology*) *inverse*

begin

definition *inverse_interval* :: 'a *interval* \Rightarrow 'a *interval*

where *inverse_interval* a = *mk_interval* (1 / (upper a), 1 / (lower a))

definition *divide_interval* :: 'a *interval* \Rightarrow 'a *interval* \Rightarrow 'a *interval*

where *divide_interval* a b = *inverse* b * a

instance *<proof>*

end

interpretation *interval_division_inverse* *divide* *inverse*

<proof>

lemma *width_of_reciprocal_interval_bound_real*:

fixes *Y* :: *real interval*

assumes $\langle \neg 0 \in_i Y \rangle$

shows $\langle \text{interval_of}(\text{width}((\text{interval_of } 1) / Y)) \leq$

$(\text{abs_interval}((\text{interval_of } 1) / Y) * \text{abs_interval}((\text{interval_of } 1) / Y)) * \text{interval_of}(\text{width } Y) \rangle$

<proof>

end

5 Affine Functions (📄 Affine_Functions)

In this theory, we provide formalisation of affine functions (sometimes also called linear polynomial functions).

theory

Affine_Functions

imports

Complex_Main

begin

5.1 Definition of Affine Functions, Alternative Definitions, and Special Cases

locale *affine_fun* =

fixes *f*

assumes $\langle \exists b. \text{linear } (\lambda x. f x - b) \rangle$

lemma *affine_fun_alt*:

$\langle \text{affine_fun } f = (\exists c g. (f = (\lambda x. g x + c)) \wedge \text{linear } g) \rangle$

$\langle \text{proof} \rangle$

lemma *affine_fun_real_linfun*:

$\langle \text{affine_fun } (f :: (\text{real} \Rightarrow \text{real})) = (\exists a b. f = (\lambda x. a * x + b)) \rangle$

$\langle \text{proof} \rangle$

lemma *linear_is_affine_fun*: $\langle \text{linear } f \Longrightarrow \text{affine_fun } f \rangle$

$\langle \text{proof} \rangle$

lemma *affine_zero_is_linear*: **assumes** $\langle \text{affine_fun } f \rangle$ **and** $\langle f 0 = 0 \rangle$ **shows** $\langle \text{linear } f \rangle$

$\langle \text{proof} \rangle$

lemma *affine_add*:

assumes $\langle \text{affine_fun } f \rangle$ **and** $\langle \text{affine_fun } g \rangle$

shows $\langle \text{affine_fun } (\lambda x. f x + g x) \rangle$

$\langle \text{proof} \rangle$

lemma *scaleR*:

assumes $\langle \text{affine_fun } f \rangle$ **shows** $\langle \text{affine_fun } (\lambda x. a *_R (f x)) \rangle$

$\langle \text{proof} \rangle$

lemma *real_affine_funD*:

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

assumes *affine_fun* *f* **obtains** *c b* **where** $f = (\lambda x. c * x + b)$

$\langle \text{proof} \rangle$

5.2 Common Linear Polynomial Functions

lemma *affine_fun_const[simp]*: $\langle \text{affine_fun } (\lambda x. c) \rangle$

$\langle \text{proof} \rangle$

lemma *affine_fun_id*[simp]: $\langle \text{affine_fun } (\lambda x. x) \rangle$
<proof>

lemma *affine_fun_mult*[simp]: $\langle \text{affine_fun } (\lambda x. (c::'a::\text{real_algebra}) * x) \rangle$
<proof>

lemma *affine_fun_scaled*[simp]: $\langle \text{affine_fun } (\lambda x. x / c) \rangle$
for $c :: 'a::\text{real_normed_field}$
<proof>

lemma *affine_fun_add*[simp]: $\langle \text{affine_fun } (\lambda x. x + c) \rangle$
<proof>

lemma *affine_fun_diff*[simp]: $\langle \text{affine_fun } (\lambda x. x - c) \rangle$
<proof>

lemma *affine_fun_triv*[simp]: $\langle \text{affine_fun } (\lambda x. a *_R x + c) \rangle$
<proof>

lemma *affine_fun_add_const*[simp]: **assumes** $\langle \text{affine_fun } f \rangle$ **shows** $\langle \text{affine_fun } (\lambda x. (f x) + c) \rangle$
<proof>

lemma *affine_fun_diff_const*[simp]: **assumes** $\langle \text{affine_fun } f \rangle$ **shows** $\langle \text{affine_fun } (\lambda x. (f x) - c) \rangle$
<proof>

lemma *affine_fun_comp*[simp]: **assumes** $\langle \text{affine_fun } (f) \rangle$
and $\langle \text{affine_fun } (g) \rangle$ **shows** $\langle \text{affine_fun } (f \circ g) \rangle$
<proof>

lemma *affine_fun_linear*[simp]: **assumes** $\langle \text{affine_fun } f \rangle$ **shows** $\langle \text{affine_fun } (\lambda x. a *_R (f x) + c) \rangle$
<proof>

5.3 Linear Polynomial Functions and Orderings

lemma *affine_fun_leq_pos*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ **and** $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{0..u\} \rangle$ **and** $\langle f 0 \leq g 0 \rangle$ **and** $\langle f u \leq g u \rangle$
shows $\langle f x \leq g x \rangle$
<proof>

lemma *affine_fun_leq_neg*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ **and** $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{l..0\} \rangle$ **and** $\langle f l \leq g l \rangle$ **and** $\langle f 0 \leq g 0 \rangle$
shows $\langle f x \leq g x \rangle$
<proof>

lemma *affine_fun_leq*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ **and** $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{l..u\} \rangle$ **and** $\langle f l \leq g l \rangle$ **and** $\langle f u \leq g u \rangle$
shows $\langle f x \leq g (x::\text{real}) \rangle$
<proof>

lemma *affine_fun_le_pos*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ and $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{o..u\} \rangle$ and $\langle f\ o < g\ o \rangle$ and $\langle f\ u < g\ u \rangle$
shows $\langle f\ x < g\ (x::\text{real}) \rangle$
\langle proof \rangle

lemma *affine_fun_le_neg*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ and $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{l..o\} \rangle$ and $\langle f\ l < g\ l \rangle$ and $\langle f\ o < g\ o \rangle$
shows $\langle f\ x < g\ (x::\text{real}) \rangle$
\langle proof \rangle

lemma *affine_fun_le*:
assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$ and $\langle \text{affine_fun } g \rangle$
and $\langle x \in \{l..u\} \rangle$ and $\langle f\ l < g\ l \rangle$ and $\langle f\ u < g\ u \rangle$
shows $\langle f\ x < g\ (x::\text{real}) \rangle$
\langle proof \rangle

end

6 Interval Inclusion Isotonicity (📄 Inclusion_Isotonicity)

theory

Inclusion_Isotonicity

imports

Interval_Utilities

Affine_Functions

Interval_Division_Non_Zero

begin

6.1 Interval Extension

6.1.1 Textbook Definition of Interval Extension

definition

$is_interval_extension_of :: \langle 'a::preorder\ interval \Rightarrow 'b::preorder\ interval \rangle \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
 $(infix (is_interval_extension_of) 50)$

where

$\langle (F\ is_interval_extension_of\ f) \rangle = (\forall x. (F\ (interval_of\ x)) = interval_of\ (f\ x))$

definition $\langle extend_natural\ f = (\lambda X. mk_interval\ (f\ (lower\ X), f\ (upper\ X))) \rangle$

lemma *interval_extension_comp*:

assumes *interval_ext_F*: $\langle F\ is_interval_extension_of\ f \rangle$

and *interval_ext_G*: $\langle G\ is_interval_extension_of\ g \rangle$

shows $\langle (F\ o\ G)\ is_interval_extension_of\ (f\ o\ g) \rangle$

<proof>

lemma *interval_extension_const*: $\langle (\lambda x. interval_of\ c)\ is_interval_extension_of\ (\lambda x. c) \rangle$

<proof>

lemma *interval_extension_id*: $\langle (\lambda x. x)\ is_interval_extension_of\ (\lambda x. x) \rangle$

<proof>

6.1.2 A Stronger Definition of Interval Extension

definition

$is_natural_interval_extension_of :: \langle 'a::linorder\ interval \Rightarrow 'b::linorder\ interval \rangle \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
 $(infix (is_natural_interval_extension_of) 50)$

where

$\langle (F\ is_natural_interval_extension_of\ f) \rangle = (\forall l\ u. (F\ (mk_interval\ (l,u))) = mk_interval\ (f\ l, f\ u))$

lemma $\langle (extend_natural\ f)\ is_interval_extension_of\ f \rangle$

<proof>

lemma $\langle (extend_natural\ f)\ is_natural_interval_extension_of\ f \rangle$

<proof>

lemma *natural_interval_extension_implies_interval_extension*:

assumes $\langle F \text{ is_natural_interval_extension_of } f \rangle$ **shows** $\langle F \text{ is_interval_extension_of } f \rangle$
 $\langle \text{proof} \rangle$

lemma *const_add_is_natural_interval_extensions*:
 $\langle (\lambda x. (\text{interval_of } c) + x) \text{ is_natural_interval_extension_of } (\lambda x. c + x) \rangle$
 $\langle \text{proof} \rangle$

lemma *const_times_is_natural_interval_extensions*:
 $\langle (\lambda x. (\text{interval_of } c) * x) \text{ is_natural_interval_extension_of } (\lambda x. c * x) \rangle$
 $\langle \text{proof} \rangle$

lemma *const_is_interval_extension*: $\langle F \text{ is_natural_interval_extension_of } (\lambda x. b) \implies F = (\lambda x. (\text{interval_of } b)) \rangle$
 $\langle \text{proof} \rangle$

lemma *id_is_interval_extension*: $\langle F \text{ is_natural_interval_extension_of } (\lambda x. x) \implies F = (\lambda x. x) \rangle$
 $\langle \text{proof} \rangle$

lemma *extend_natural_is_interval_extension*:
 $\langle (\text{extend_natural } f) \text{ is_natural_interval_extension_of } f \rangle$
 $\langle \text{proof} \rangle$

lemma *is_natural_interval_extension_implies_bounds*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\langle F \text{ is_natural_interval_extension_of } f \rangle$ **and** $\langle F x \leq F x' \rangle$
shows
 $\langle ((f (\text{lower } x')) \leq (f (\text{lower } x))) \vee ((f (\text{upper } x')) \leq (f (\text{upper } x))) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_lower*:
 $\langle ((F \text{ is_interval_extension_of } f) \implies \text{lower } (F (\text{interval_of } x)) = (f x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_upper*:
 $\langle ((F \text{ is_interval_extension_of } f) \implies \text{upper } (F (\text{interval_of } x)) = (f x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *is_interval_extension_eq_upper_and_lower*:
 $\langle ((F \text{ is_interval_extension_of } f) \implies (\forall x. \text{lower } (F (\text{interval_of } x)) = (f x) \wedge \text{upper } (F (\text{interval_of } x)) = (f x))) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_lower_simp[simp]*:
assumes $\langle F \text{ is_interval_extension_of } f \rangle$ **and** $\langle X = \text{Interval}(x, x) \rangle$
shows $\langle \text{lower } (F X) = f x \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_upper_simp[simp]*:
assumes $\langle F \text{ is_interval_extension_of } f \rangle$ **and** $\langle X = \text{Interval}(x, x) \rangle$
shows $\langle \text{upper } (F X) = f x \rangle$
 $\langle \text{proof} \rangle$

6.2 Interval Inclusion Isotonicity

In this section, we introduce the concept of inclusion isotonicity. The formalization in this theory generalises the definitions from [5]:

definition

$\text{inclusion_isotonic} :: \langle ('a::\text{preorder interval} \Rightarrow 'b::\text{preorder interval}) \Rightarrow \text{bool} \rangle$

where

$\langle \text{inclusion_isotonic } f = (\forall x x'. x \leq x' \longrightarrow (f x) \leq (f x')) \rangle$

We can immediately show that any natural extension of an affine function of from type *real* to *real* is interval inclusion isotonic:

lemma *real_affine_fun_is_inclusion_isotonicM*:

assumes $\langle \text{affine_fun } (f::\text{real} \Rightarrow \text{real}) \rangle$

shows $\langle \text{inclusion_isotonic } (\text{extend_natural } f) \rangle$

$\langle \text{proof} \rangle$

definition

$\text{inclusion_isotonic_on} :: \langle ('a \text{ interval} \Rightarrow \text{bool}) \Rightarrow ('a::\text{preorder interval} \Rightarrow 'b::\text{preorder interval}) \Rightarrow \text{bool} \rangle$

where

$\langle \text{inclusion_isotonic_on } P f = (\forall x x'. P x \wedge P x' \wedge x \leq x' \longrightarrow (f x) \leq (f x')) \rangle$

lemma *inclusion_isotonic_eq*: $\langle \text{inclusion_isotonic_on } (\lambda x. \text{True}) = \text{inclusion_isotonic} \rangle$

$\langle \text{proof} \rangle$

definition *inclusion_isotonic_binary* :: $\langle ('a::\text{preorder interval} \Rightarrow 'a \text{ interval} \Rightarrow 'b::\text{preorder interval}) \Rightarrow \text{bool} \rangle$

where

$\langle \text{inclusion_isotonic_binary } f = (\forall x x' y y'. x \leq x' \wedge y \leq y' \longrightarrow (f x y) \leq (f x' y')) \rangle$

definition *inclusion_isotonic_binary_on* :: $\langle ('a \text{ interval} \Rightarrow \text{bool}) \Rightarrow ('a::\text{preorder interval} \Rightarrow 'a \text{ interval} \Rightarrow 'b::\text{preorder interval}) \Rightarrow \text{bool} \rangle$

where

$\langle \text{inclusion_isotonic_binary_on } P f = (\forall x x' y y'. P x \wedge P x' \wedge P y \wedge P y' \wedge x \leq x' \wedge y \leq y' \longrightarrow (f x y) \leq (f x' y')) \rangle$

lemma *inclusion_isotonic_binary_eq*: $\langle \text{inclusion_isotonic_binary_on } (\lambda x. \text{True}) = \text{inclusion_isotonic_binary} \rangle$

$\langle \text{proof} \rangle$

definition *inclusion_isotonicM_n* :: $\langle \text{nat} \Rightarrow ('a::\text{linorder interval list} \Rightarrow 'a::\text{linorder interval}) \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{inclusion_isotonicM_n } n f = (\forall \text{ is js. } (\text{length is} = n \wedge \text{length js} = n \wedge (\text{is} \leq_I \text{js})) \longrightarrow f \text{ is} \leq f \text{ js}) \rangle$

definition *inclusion_isotonicM_n_on* :: $\langle ('a \text{ interval} \Rightarrow \text{bool}) \Rightarrow \text{nat} \Rightarrow ('a::\text{linorder interval list} \Rightarrow 'a::\text{linorder interval}) \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{inclusion_isotonicM_n_on } P n f = (\forall \text{ is js. } (\forall i \in \text{set is. } P i) \wedge (\forall j \in \text{set js. } P j) \wedge (\text{length is} = n \wedge \text{length js} = n \wedge (\text{is} \leq_I \text{js})) \longrightarrow f \text{ is} \leq f \text{ js}) \rangle$

lemma *inclusion_isotonicM_n_eq*: $\langle \text{inclusion_isotonicM_n_on } (\lambda x. \text{True}) = \text{inclusion_isotonicM_n} \rangle$

$\langle \text{proof} \rangle$

Finally, we extend the definition to functions over lists and show that the three definitions of interval inclusion isotonicity are, for their corresponding types, equivalent:

locale *inclusion_isotonicM* =

fixes $n :: \text{nat}$

and $f :: \langle ('a::\text{linorder}) \text{ interval list} \Rightarrow 'a \text{ interval list} \rangle$

```

assumes inclusion_isotonicM :
  ⟨(∀ is js. (length is = n) ∧ (length js = n) ∧ (is ≤I js) ⟶ f is ≤I f js)⟩
begin
end

locale inclusion_isotonicM_on =
  fixes P :: ⟨'a::linorder interval ⇒ bool⟩
    and n :: nat
    and f :: ⟨('a::linorder) interval list ⇒ 'a interval list⟩
  assumes inclusion_isotonicM :
  ⟨(∀ is js. (∀ i ∈ set is. P i) ∧ (∀ j ∈ set js. P j) ∧ (length is = n) ∧ (length js = n) ∧ (is ≤I js) ⟶ f is ≤I f js)⟩
begin
end

lemma inclusion_isotonicM_on_eq: ⟨inclusion_isotonicM_on (λ x. True) = inclusion_isotonicM⟩
  ⟨proof⟩

lemma inclusion_isotonic_conv:
  ⟨inclusion_isotonic g = inclusion_isotonicM 1 (λ xs . case xs of [x] ⇒ [g x])⟩
  ⟨proof⟩

lemma inclusion_isotonicM_n_conv1:
  ⟨inclusion_isotonicM n f = inclusion_isotonicM n (λ xs. [f xs])⟩
  ⟨proof⟩

lemma inclusion_isotonicM_conv2:
  assumes ⟨inclusion_isotonicM n f⟩
  and ⟨∀ xs. (length xs = n) ⟶ N = (length (f xs))⟩
  shows ⟨inclusion_isotonicM n (λ xs. if n' < N then [(f xs)!n'] else [])⟩
  ⟨proof⟩

lemma inclusion_isotonicM_n_on_conv1:
  ⟨inclusion_isotonicM_n_on P n f = inclusion_isotonicM_on P n (λ xs. [f xs])⟩
  ⟨proof⟩

lemma inclusion_isotonicM_on_conv2:
  assumes ⟨inclusion_isotonicM_on P n f⟩
  and ⟨∀ xs. (length xs = n) ⟶ N = (length (f xs))⟩
  shows ⟨inclusion_isotonicM_on P n (λ xs. if n' < N then [(f xs)!n'] else [])⟩
  ⟨proof⟩

lemma inclusion_isotonic_binary_conv1:
  ⟨inclusion_isotonic_binary f = inclusion_isotonicM_n 2 (λ xs . case xs of [x,y] ⇒ f x y)⟩
  ⟨proof⟩

lemma inclusion_isotonic_binary_conv2:
  ⟨inclusion_isotonic_binary f = inclusion_isotonicM 2 (λ xs . case xs of [x,y] ⇒ [f x y])⟩
  ⟨proof⟩

lemma inclusion_isotonic_binary_on_conv1:
  ⟨inclusion_isotonic_binary_on P f = inclusion_isotonicM_n_on P 2 (λ xs . case xs of [x,y] ⇒ f x y)⟩
  ⟨proof⟩

lemma inclusion_isotonic_binary_on_conv2:

```


$\langle \text{inclusion_isotonic_binary_on } P f = \text{inclusion_isotonicM_on } P 2 (\lambda xs . \text{case } xs \text{ of } [x,y] \Rightarrow [f x y]) \rangle$
 $\langle \text{proof} \rangle$

6.2.1 Compositionality of Interval Inclusion Isotonicity

lemma *inclusion_isotonicM_comp*:
assumes $\langle \text{inclusion_isotonicM } n f \rangle$
and $\langle \text{inclusion_isotonicM } m g \rangle$
and $\langle \forall xs . \text{length } xs = n \longrightarrow \text{length } (f xs) = m \rangle$
shows $\langle \text{inclusion_isotonicM } n (g \circ f) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonicM_on_comp*:
assumes $\langle \text{inclusion_isotonicM_on } P n f \rangle$
and $\langle \text{inclusion_isotonicM_on } Q m g \rangle$
and $\langle \forall xs . \text{length } xs = n \longrightarrow \text{length } (f xs) = m \rangle$
and $\langle \forall is . (\forall i \in \text{set } is . P i) \longrightarrow (\forall x \in \text{set } (f is) . Q x) \rangle$
shows $\langle \text{inclusion_isotonicM_on } P n (g \circ f) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonic_comp*:
assumes $\langle \text{inclusion_isotonic } f \rangle$
and $\langle \text{inclusion_isotonic } g \rangle$
shows $\langle \text{inclusion_isotonic } (g \circ f) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonic_on_comp*:
assumes $\langle \text{inclusion_isotonic_on } P f \rangle$
and $\langle \text{inclusion_isotonic_on } Q g \rangle$
and $\langle \forall x . P x \longrightarrow Q (f x) \rangle$
shows $\langle \text{inclusion_isotonic_on } P (g \circ f) \rangle$
 $\langle \text{proof} \rangle$

6.2.2 Interval Inclusion Isotonicity of the Core Operator

Unary Minus (Negation)

lemma *inclusion_isotonicM_uminus[simp]*: $\langle \text{inclusion_isotonic } \text{uminus} \rangle$
 $\langle \text{proof} \rangle$

Addition

lemma *inclusion_isotonicM_plus[simp]*: $\langle \text{inclusion_isotonic_binary } (+) \rangle$
 $\langle \text{proof} \rangle$

Substraction

lemma *inclusion_isotonicM_minus[simp]*: $\langle \text{inclusion_isotonic_binary } (-) \rangle$
 $\langle \text{proof} \rangle$

Multiplication

lemma *inclusion_isotonicM_times[simp]*:
 $\langle \text{inclusion_isotonic_binary } (\lambda x y . (x :: 'a :: \{\text{linordered_ring, real_normed_algebra, linear_continuum_topology}\} \text{interval}) * y) \rangle$

<proof>

6.2.3 Interval Inclusion Isotonicity of Various Functions

lemma *inclusion_isotonicM_n_empty*[simp]: $\langle \text{inclusion_isotonicM } n \ (\lambda \text{ xs. []}) \rangle$
<proof>

lemma *inclusion_isotonic_id*[simp]: $\langle \text{inclusion_isotonic id} \rangle$
<proof>

lemma *inclusion_isotonicM_id*[simp]: $\langle \text{inclusion_isotonicM } n \ \text{id} \rangle$
<proof>

lemma *inclusion_isotonicM_hd*[simp]:
assumes $\langle 0 < n \rangle$
shows $\langle \text{inclusion_isotonicM } n \ \text{hd} \rangle$
<proof>

lemma *inclusion_isotonic_add_const1*[simp]:
 $\langle \text{inclusion_isotonic } (\lambda x. x + c) \rangle$
<proof>

lemma *inclusion_isotonicM_1_add_const2*[simp]:
 $\langle \text{inclusion_isotonic } (\lambda x. c + x) \rangle$
<proof>

lemma *inclusion_isotonic_times_right*[simp]:
 $\langle \text{inclusion_isotonic } (\lambda x. C * (x :: 'a :: \text{linordered_ring interval})) \rangle$
<proof>

lemma *inclusion_isotonic_times_left*[simp]:
 $\langle \text{inclusion_isotonic } (\lambda x. (x :: 'a :: \{\text{real_normed_algebra, linordered_ring, linear_continuum_topology}\} \text{interval}) * C) \rangle$
<proof>

lemma *map_inclusion_isotonicM*:
assumes $\langle \text{inclusion_isotonic } f \rangle$
shows $\langle \text{inclusion_isotonicM } n \ (\text{map } f) \rangle$
<proof>

lemma *inclusion_isotonicM_fun_plus*:
assumes $\langle \text{inclusion_isotonic } f \rangle$ and $\langle \text{inclusion_isotonic } g \rangle$
shows $\langle \text{inclusion_isotonic } (\lambda x. f x + g x) \rangle$
<proof>

lemma *inclusion_isotonic_plus_const*:
assumes $\langle \text{inclusion_isotonic } f \rangle$ and $\langle \text{inclusion_isotonic } g \rangle$
shows $\langle \text{inclusion_isotonic } (\lambda x. g x + c) \rangle$
<proof>

lemma *inclusion_isotonic_times_const_real*:
assumes $\langle \text{inclusion_isotonic } f \rangle$
shows $\langle \text{inclusion_isotonic } (\lambda x. (c :: \text{real interval}) * (f x)) \rangle$
<proof>

lemma *intervall_le_foldr*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \text{length } js = \text{length } is \implies is \leq_I js \implies (\text{foldr } f \text{ is } e) \leq (\text{foldr } f \text{ js } e) \rangle$

$\langle \text{proof} \rangle$

lemma *intervall_le_foldr_map*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

and $\langle \text{inclusion_isotonic } g \rangle$

shows $\langle \text{length } js = \text{length } is \implies is \leq_I js \implies (\text{foldr } f (\text{map } g \text{ is}) e) \leq (\text{foldr } f (\text{map } g \text{ js}) e) \rangle$

$\langle \text{proof} \rangle$

lemma *intervall_le_foldr_e*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

and $\langle is \leq js \rangle$

shows $\langle (\text{foldr } f \text{ xs } is) \leq (\text{foldr } f \text{ xs } js) \rangle$

$\langle \text{proof} \rangle$

lemma *foldr_inclusion_isotonicM_e*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \forall x. \text{inclusion_isotonic } (\text{foldr } f \text{ x}) \rangle$

$\langle \text{proof} \rangle$

lemma *foldr_inclusion_isotonicM*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \text{inclusion_isotonicM_n } n (\lambda x. \text{foldr } f \text{ x } e) \rangle$

$\langle \text{proof} \rangle$

lemma *foldr_inclusion_isotonicM_g*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

and $\langle \text{inclusion_isotonicM } n \text{ g} \rangle$

shows $\langle \text{inclusion_isotonicM_n } n (\lambda x. \text{foldr } f ((g \text{ x})) e) \rangle$

$\langle \text{proof} \rangle$

lemma *foldr_map_inclusion_isotonicM_g*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

and $\langle \text{inclusion_isotonic } g \rangle$

and $\langle \text{inclusion_isotonicM } n \text{ P} \rangle$

shows $\langle \text{inclusion_isotonicM_n } n (\lambda x. \text{foldr } f (\text{map } g (\text{P } x)) e) \rangle$

$\langle \text{proof} \rangle$

lemma *foldl_inclusion_isotonicM*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \text{inclusion_isotonicM_n } n (\text{foldl } f \text{ e}) \rangle$

$\langle \text{proof} \rangle$

lemma *fold_inclusion_isotonicM*:

assumes $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \text{inclusion_isotonicM_n } n (\lambda x. \text{fold } f \text{ x } e) \rangle$

$\langle \text{proof} \rangle$

lemma *map2_inclusion_isotonicM_left*: **assumes** $\langle \text{inclusion_isotonic_binary } f \rangle$

shows $\langle \text{inclusion_isotonicM } n (\text{map2 } f \text{ xs}) \rangle$

$\langle \text{proof} \rangle$

lemma *map2_inclusion_isotonicM_right*: **assumes** $\langle \text{inclusion_isotonic_binary } f \rangle$
shows $\langle \text{inclusion_isotonicM } n \ (\lambda \text{ ys. } \text{map2 } f \text{ ys } \text{xs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *map2_inclusion_isotonicM_right_g*:
assumes $\langle \text{inclusion_isotonic_binary } f \rangle$
and $\langle \forall \text{ xs. } \text{length } (g \text{ xs}) = \text{length } \text{xs} \rangle$
and $\langle \text{inclusion_isotonicM } n \ g \rangle$
and $\langle \text{length } \text{xs} = n \rangle$
and $\langle \text{length } \text{is} = n \rangle$
and $\langle \text{length } \text{js} = n \rangle$
and $\langle \text{is} \leq_I \text{js} \rangle$
shows $\langle \text{map2 } f \ (g \ \text{is}) \ (h \ \text{xs}) \leq_I \text{map2 } f \ (g \ \text{js}) \ (h \ \text{xs}) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonicM_map*:
assumes $\langle \forall x \in \text{set } \text{xs} . g \ x \leq h \ x \rangle$
shows $\langle (\text{map } g \ \text{xs}) \leq_I (\text{map } h \ \text{xs}) \rangle$
 $\langle \text{proof} \rangle$

6.3 Interval Extension and Inclusion Properties

lemma *interval_extension_inclusion*:
assumes $\langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \forall X . \text{interval_of } (f \ X) \leq F \ (\text{interval_of } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_subset_const*:
assumes *interval_ext_F*: $\langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \forall X . \text{set_of } (\text{interval_of } (f \ X)) \subseteq \text{set_of } (F \ (\text{interval_of } X)) \rangle$
 $\langle \text{proof} \rangle$

lemma *fundamental_theorem_of_interval_analysis*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes *interval_ext_F*: $\langle F \text{ is_interval_extension_of } f \rangle$
and *inc_isotonic_F*: $\langle \text{inclusion_isotonic } F \rangle$
shows $\langle \forall X . f' \ (\text{set_of } X) \subseteq \text{set_of } (F \ X) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_extension_bounds*:
assumes *inclusion_isotonic F*
and *F is_interval_extension_of f*
shows $\langle (\forall x \in \text{set_of } X . \text{lower } (F \ X) \leq f \ x) \vee (\forall x \in \text{set_of } X . f \ x \leq \text{lower } (F \ X)) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonic_extension_subset*:
assumes *inclusion_isotonic F*
and *F is_interval_extension_of f*
shows $\langle f' \ \text{set_of } X \subseteq \text{set_of } (F \ X) \rangle$
 $\langle \text{proof} \rangle$

lemma *inclusion_isotonic_extension_includes*:

assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \forall x \in \text{set_of } X. f\ x \in \text{set_of } (F\ X) \rangle$
<proof>

lemma *inclusion_isotonic_extension_lower_bound*:

assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \forall x \in \text{set_of } X. \text{lower } (F\ X) \leq f\ x \rangle$
<proof>

lemma *inclusion_isotonic_extension_upper_bound*:

assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \forall x \in \text{set_of } X. f\ x \leq \text{upper } (F\ X) \rangle$
<proof>

lemma *inclusion_isotonic_inf*:

fixes *F::<real interval \Rightarrow real interval>*
assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \text{lower } (F\ (X::\text{real interval})) \leq \text{Inf } (f\ \text{'set_of } X) \rangle$
<proof>

lemma *inclusion_isotonic_sup*:

fixes *F::<real interval \Rightarrow real interval>*
assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \text{Sup } (f\ \text{'set_of } X) \leq \text{upper } (F\ X) \rangle$
<proof>

lemma *lower_inf*:

fixes *F::<real interval \Rightarrow real interval>*
assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \text{Inf } (f\ \text{'set_of } X) \leq f\ (\text{lower } X) \rangle$
<proof>

lemma *upper_sup*:

fixes *F::<real interval \Rightarrow real interval>*
assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle f\ (\text{upper } X) \leq \text{Sup } (f\ \text{'set_of } X) \rangle$
<proof>

lemma *lower_F_f*:

fixes *F::<real interval \Rightarrow real interval>*
assumes *inclusion_isotonic* *F*
and *F is_interval_extension_of f*
shows $\langle \text{lower } (F\ X) \leq f\ (\text{lower } X) \rangle$
<proof>

lemma *upper_F_f*:

```

fixes F::⟨real interval ⇒ real interval⟩
assumes inclusion_isotonic F
and F is_interval_extension_of f
shows f (upper X) ≤ upper (F X)
⟨proof⟩

```

```

lemma inclusion_isotonic_interval_extension_le:
  assumes inclusion_isotonic: ⟨inclusion_isotonic F⟩
  and interval_extension: ⟨F is_interval_extension_of f⟩
  and adjacent: ⟨upper a = lower b⟩
shows ⟨lower (F b) ≤ upper (F a)⟩
⟨proof⟩

```

6.4 Division

```

context interval_division_inverse
begin
lemma inclusion_isotonic_on_inverse[simp]:
  ⟨inclusion_isotonic_on (λ x . ¬ 0 ∈i x) ((inverse::('a interval ⇒ 'a interval)))⟩
  ⟨proof⟩

lemma inclusion_isotonic_on_division[simp]:
  ⟨inclusion_isotonic_binary_on (λ x . ¬ 0 ∈i x) (λ x y . divide x y)⟩
  ⟨proof⟩

end

end

```

7 Lipschitz Continuity of Intervals

(Lipschitz_Interval_Extension)

An extension of of Lipschitz Continuity, developed for the Lipschitz Continuity of intervals.

theory

Lipschitz_Interval_Extension

imports

Inclusion_Isotonicity

HOL-Analysis.Lipschitz

Interval_Utilities

begin

7.1 Definition of Lipschitz Continuity on Intervals

An interval extension F is said to be lipschitz in X if there is a constant L such that $w(F X) \leq L w X$ for every $X \subseteq X$. Hence the width of $F X$ approaches 0 at least linearly with the width of X .

definition *lipschitzl_on* :: 'a:: {zero,minus,preorder,times} \Rightarrow 'a interval set \Rightarrow ('a interval \Rightarrow 'a interval) \Rightarrow bool
where *lipschitzl_on* C U F \longleftrightarrow ($0 \leq C \wedge (\forall x \in U. \text{width}(F x) \leq C * \text{width } x)$)

The definition *lipschitzl_on* refers to definition 6.1 in [4]

bundle *lipschitzl_syntax* **begin**

notation *lipschitzl_on* ($_$ -*lipschitzl'_on* [1000])

end

bundle *no_lipschitzl_syntax* **begin**

no_notation *lipschitzl_on* ($_$ -*lipschitzl'_on* [1000])

end

unbundle *lipschitzl_syntax*

lemma *lipschitzl_onI*: C-*lipschitzl_on* U F

if $\bigwedge x. x \in U \implies \text{width}(F x) \leq C * \text{width } x$ $0 \leq C$

<proof>

lemma *lipschitzl_onD*:

$\text{width}(F x) \leq C * \text{width } x$

if C-*lipschitzl_on* U F $x \in U$

<proof>

lemma *lipschitzl_on_nonneg*:

$0 \leq C$ **if** C-*lipschitzl_on* U F

<proof>

lemma *lipschitz_on_normD*:

$\text{norm}(\text{width}(F x)) \leq C * \text{norm}(\text{width } x)$

if C-*lipschitzl_on* U F $x \in U$

<proof>

lemma *lipschitzl_on_mono*: $L\text{-lipschitzl_on } D (F:: 'a::\{\text{linordered_ring}\} \text{ interval} \Rightarrow 'a \text{ interval})$
if $M\text{-lipschitzl_on } E F M \leq L D \subseteq E$
<proof>

lemmas *lipschitzl_on_subset*
 $= \text{lipschitzl_on_mono}[OF_ _ \text{order_refl}]$
and *lipschitzl_on_le* $= \text{lipschitzl_on_mono}[OF_ _ \text{order_refl}]$

lemma *lipschitzl_on_lel*:
 $C\text{-lipschitzl_on } U F$
if $\bigwedge x. x \in U \Rightarrow \text{width } (F x) \leq C * \text{width } x$
 $0 \leq C$
for $F:: 'a::\{\text{minus,preorder,times,zero}\} \text{ interval} \Rightarrow 'a \text{ interval}$
<proof>

7.1.1 Lipschitz Continuity of Operations

Let F and G be inclusion isotonic interval extensions with F Lipschitz in Y and G Lipschitz in X and $G X \subseteq Y$. Then the composition $H X = F (G X)$ is Lipschitz in X and is inclusion isotonic

lemma *lipschitzl_on_compose*:
fixes $U :: \langle 'a::\{\text{linordered_ring}\} \text{ interval set} \rangle$
assumes $f: C\text{-lipschitzl_on } U F$ **and** $g: D\text{-lipschitzl_on } (F'U) G$
shows $(D * C)\text{-lipschitzl_on } U (G \circ F)$
<proof>

The definition $\llbracket ?C\text{-lipschitzl_on } ?U ?F; ?D\text{-lipschitzl_on } (?F' ?U) ?G \rrbracket \Longrightarrow (?D * ?C)\text{-lipschitzl_on } ?U (?G \circ ?F)$ refers to lemma 6.3 in [4]

lemma *lipschitzl_on_compose2*:
fixes $U :: \langle 'a::\{\text{linordered_ring}\} \text{ interval set} \rangle$
assumes $F: C\text{-lipschitzl_on } U F$ **and** $G: D\text{-lipschitzl_on } (F'U) G$
shows $(D * C)\text{-lipschitzl_on } U (\lambda x. G (F x))$
<proof>

lemma *lipschitzl_on_cong*:
 $C\text{-lipschitzl_on } U G \longleftrightarrow D\text{-lipschitzl_on } V F$
if $C = D U = V \bigwedge x. x \in V \Rightarrow G x = F x$
<proof>

lemma *lipschitzl_on_transform*:
 $C\text{-lipschitzl_on } U G$
if $C\text{-lipschitzl_on } U F$
 $\bigwedge x. x \in U \Rightarrow G x = F x$
<proof>

lemma *lipschitz_on_empty_iff*: $C\text{-lipschitzl_on } \{\} f \longleftrightarrow C \geq 0$
<proof>

lemma *lipschitz_on_id*: $(1::\text{real})\text{-lipschitzl_on } U (\lambda x. x)$
<proof>

lemma *lipschitz_on_constant*:
assumes $\langle \text{lower } c = \text{upper } c \rangle$

shows $(o::\text{real})\text{-lipschitz_on } U (\lambda x. c)$
<proof>

lemma *lipschitz_on_mult*:
fixes $X :: 'a::\{\text{linordered_idom}\}$
assumes *lipschitz_on C U f*
and $1 \leq X$
shows *lipschitz_on (X*C) U f*
<proof>

7.1.2 Interval bounds on reals

lemma *bounded_image_real*:
fixes $X :: \text{real interval}$ **and** $f :: \text{real} \Rightarrow \text{real}$
assumes $\forall x \in \{\text{lower } X.. \text{upper } X\}.$
 $f (\text{lower } X) - L * (\text{upper } X - \text{lower } X) \leq f x \wedge f x \leq f (\text{lower } X) + L * (\text{upper } X - \text{lower } X)$
shows $\exists x e. \forall y \in f' \{\text{lower } X.. \text{upper } X\}. \text{dist } x y \leq e$
<proof>

lemma *lipschitz_bounded_image_real*:
fixes $X :: \text{real set}$ **and** $f :: \text{real} \Rightarrow \text{real}$
assumes *bounded X* **and** $L\text{-lipschitz_on } X f$
shows *bounded (f' X)*
<proof>

lemma *inf_le_sup_image_real*:
fixes $X :: \text{real interval}$ **and** $f :: \text{real} \Rightarrow \text{real}$
assumes $L\text{-lipschitz_on } (\text{set_of } X) f$
shows $\text{Inf } (f' \text{set_of } X) \leq \text{Sup } (f' \text{set_of } X)$
<proof>

lemma *sup_image_le_real*:
fixes $f :: \text{real} \Rightarrow \text{real}$ **and** $F :: \text{real interval} \Rightarrow \text{real interval}$ **and** $X :: \text{real interval}$
assumes $f' \text{set_of } X \subseteq \text{set_of } (F X)$
and $L\text{-lipschitz_on } (\text{set_of } X) f$
shows $\text{Sup } (f' \text{set_of } X) \leq \text{Sup } (\text{set_of } (F X))$
<proof>

lemma *inf_image_le_real*:
fixes $f :: \text{real} \Rightarrow \text{real}$ **and** $F :: \text{real interval} \Rightarrow \text{real interval}$ **and** $X :: \text{real interval}$
assumes $f' \text{set_of } X \subseteq \text{set_of } (F X)$
and $L\text{-lipschitz_on } (\text{set_of } X) f$
shows $\text{Inf } (\text{set_of } (F X)) \leq \text{Inf } (f' (\text{set_of } X))$
<proof>

end

8 Multi-Intervals (Multi_Interval_Preliminaries)

8.1 Preliminaries

theory

Multi_Interval_Preliminaries

imports

HOL-Library.Interval

HOL-Analysis.Analysis

Inclusion_Isotonicity

begin

8.1.1 A Class for Capturing Monotonicity of Minus

We try to keep our formalisation of interval arithmetic as generic as possible. In particular, we want to support intervals of type *nat*, *int*, *real*, and *ereal*. For all these types, minus (subtraction) is monotonous. Sadly, Isabelle lacks a type class capturing this. Luckily, it is very easy to define our own:

```
class minus_mono = minus + linorder +  
  assumes minus_mono:  $A \leq B \implies D \leq C \implies A - C \leq B - D$   
begin end
```

```
instance nat::minus_mono
```

<proof>

```
instance int::minus_mono
```

<proof>

```
instance real::minus_mono
```

<proof>

```
instance integer::minus_mono
```

<proof>

```
instance ereal::minus_mono
```

<proof>

8.1.2 Infrastructure for Lifting Interval Operations to Lists of Intervals

```
definition un_op_interval_list:: $\langle 'a \text{ interval} \Rightarrow 'a \text{ interval} \rangle$   
   $\Rightarrow 'a \text{ interval list} \Rightarrow 'a \text{ interval list}$ 
```

where

$\langle \text{un_op_interval_list } op \text{ } xs = \text{map } op \text{ } xs \rangle$

```
definition bin_op_interval_list:: $\langle 'a \text{ interval} \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ interval} \rangle$   
   $\Rightarrow 'a \text{ interval list} \Rightarrow 'a \text{ interval list} \Rightarrow 'a \text{ interval list}$ 
```

where

$\langle \text{bin_op_interval_list } op \text{ } xs \text{ } ys = \text{concat } (\text{map } (\lambda x . \text{map } (op \text{ } x) \text{ } xs) \text{ } ys) \rangle$

```
lemma bin_op_interval_list_non_empty:  $\langle (xs \neq [] \wedge ys \neq []) = (\text{bin\_op\_interval\_list } op \text{ } xs \text{ } ys \neq []) \rangle$   
<proof>
```

lemma *bin_op_interval_list_empty*: $\langle (xs = [] \vee ys = []) = (bin_op_interval_list\ op\ xs\ ys = []) \rangle$
 $\langle proof \rangle$

lemma *bin_op_interval_unroll*: $\langle bin_op_interval_list\ op\ (xs)\ (y\#\!ys) = (map\ (op\ y)\ xs)\ @\ (bin_op_interval_list\ op\ xs\ ys) \rangle$
 $\langle proof \rangle$

lemma *bin_op_interval_list_commute*:
assumes *op_commute*: $\langle \bigwedge x\ y.\ op\ x\ y = op\ y\ x \rangle$
shows $\langle set\ (bin_op_interval_list\ (op)\ xs\ ys) = set\ (bin_op_interval_list\ (op)\ ys\ xs) \rangle$
 $\langle proof \rangle$

lemma *bin_op_interval_list_assoc*:
assumes *op_assoc*: $\langle \bigwedge x\ y\ z.\ op\ (op\ x\ y)\ z = op\ x\ (op\ y\ z) \rangle$
shows $\langle set\ (bin_op_interval_list\ (op)\ ((bin_op_interval_list\ (op)\ xs\ ys))\ zs) = set\ (bin_op_interval_list\ (op)\ xs\ ((bin_op_interval_list\ (op)\ ys\ zs))) \rangle$
 $\langle proof \rangle$

Lifting Unary Minus, Addition, and Multiplication

definition $\langle iList_uminus = un_op_interval_list\ (\lambda x.\ -\ x) \rangle$

definition $\langle iList_plus = bin_op_interval_list\ (+) \rangle$

definition $\langle iList_times = bin_op_interval_list\ (*) \rangle$

lemma *iList_plus_lower*:
assumes $\langle A \neq [] \rangle$ and $\langle B \neq [] \rangle$
shows $\langle lower\ (hd\ (iList_plus\ A\ B)) = lower\ (hd\ A) + lower\ (hd\ B) \rangle$
 $\langle proof \rangle$

lemma *iList_plus_upper*:
assumes $\langle A \neq [] \rangle$ and $\langle B \neq [] \rangle$
shows $\langle upper\ (hd\ (iList_plus\ A\ B)) = upper\ (hd\ A) + upper\ (hd\ B) \rangle$
 $\langle proof \rangle$

lemma *iList_plus_unroll*:
 $\langle iList_plus\ ys\ (x\ \#\!xs) = map\ ((+)\ x)\ ys\ @\ iList_plus\ ys\ xs \rangle$
 $\langle proof \rangle$

lemma $a \neq [] \implies (iList_plus\ [Interval\ (o,\ o)]\ a) = (a::'a::\{ordered_ab_group_add,\ zero\}\ interval\ list)$
 $\langle proof \rangle$

lemma *iList_plus_commute*:
 $\langle set\ (iList_plus\ xs\ ys) = set\ (iList_plus\ ys\ xs) \rangle$
 $\langle proof \rangle$

lemma *iList_plus_assoc*:
 $\langle set\ (iList_plus\ xs\ (iList_plus\ ys\ zs)) = set\ (iList_plus\ (iList_plus\ xs\ ys)\ zs) \rangle$
 $\langle proof \rangle$

lemma *remdups_append1*:
 $\langle remdups\ (remdups\ xs\ @\ ys) = remdups\ (xs\ @\ ys) \rangle$
 $\langle proof \rangle$

lemma *bin_op_interval_remdups_left*:
 $\langle \text{remdups } (\text{bin_op_interval_list } \text{op } (\text{remdups } x) y) = \text{remdups } (\text{bin_op_interval_list } \text{op } x y) \rangle$
<proof>

lemma *iList_plus_remdups_left*:
 $\text{remdups } (\text{iList_plus } (\text{remdups } a) b) = \text{remdups } (\text{iList_plus } a b)$
for $a :: 'a :: \{\text{minus_mono}, \text{ordered_ab_semigroup_add}, \text{linorder}, \text{linordered_field}\}$ *interval list*
<proof>

lemma *interval_add_eq*: $\langle a + b = \text{Interval}(\text{lower } a + \text{lower } b, \text{upper } a + \text{upper } b) \rangle$
<proof>

lemma *iList_plus_lower_upper_eq*:
 $\langle \text{iList_plus } = \text{bin_op_interval_list } (\lambda a b. \text{Interval}(\text{lower } a + \text{lower } b, \text{upper } a + \text{upper } b)) \rangle$
<proof>

A Locale for Multi-Interval Division Where the Quotient does not Contain 0

context *interval_division*
begin

definition $\langle \text{iList_inverse} = \text{un_op_interval_list } \text{inverse} \rangle$
definition $\langle \text{iList_divide} = \text{bin_op_interval_list } \text{divide} \rangle$

end

8.1.3 Utilities for (Sorted) Lists of Intervals

definition $\langle \text{cmp_lower_width} = (\lambda x y. \text{if } \text{lower } x = \text{lower } y \text{ then } \text{width } x \leq \text{width } y \text{ else } \text{lower } x < \text{lower } y) \rangle$

definition $\langle \text{sorted_wrt_lower} = \text{sorted_wrt } \text{cmp_lower_width} \rangle$

lemma *interval_lower_width_eq*:
 $\langle (\text{lower } x = \text{lower } y \wedge \text{width } x = \text{width } y) = (x = (y :: 'a :: \{\text{minus_mono}, \text{linordered_field}\} \text{interval})) \rangle$
<proof>

lemma *sorted_wrt_lower_distinct_lists_eq*:
assumes $\langle \text{set } xs = \text{set } (ys :: 'a :: \{\text{minus_mono}, \text{linordered_field}\} \text{interval list}) \rangle$
and $\langle \text{distinct } xs \rangle$ **and** $\langle \text{distinct } ys \rangle$
and $\langle \text{sorted_wrt_lower } xs \rangle$ **and** $\langle \text{sorted_wrt_lower } ys \rangle$
shows $\langle xs = ys \rangle$
<proof>

definition $\langle \text{sorted_wrt_upper} = \text{sorted_wrt } (\lambda x y. \text{upper } x \leq \text{upper } y) \rangle$

definition $\langle \text{cmp_non_overlapping} = (\lambda x y. \text{upper } x \leq \text{lower } y) \rangle$

lemma *cmp_non_overlapping_lower*: $\langle \text{cmp_non_overlapping } x y \implies \text{lower } x \leq \text{lower } y \rangle$
<proof>

lemma *cmp_non_overlapping_upper*: $\langle \text{cmp_non_overlapping } x y \implies \text{upper } x \leq \text{upper } y \rangle$
<proof>

definition $\langle \text{non_overlapping_sorted} = \text{sorted_wrt } \text{cmp_non_overlapping} \rangle$

definition $\langle \text{contiguous } xs = (\forall i < \text{length } xs - 1 . \text{upper } (xs ! i) = \text{lower } (xs ! (i + 1))) \rangle$

lemma $\text{non_overlapping_sorted_empty}$: $\langle \text{non_overlapping_sorted } [] \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{non_overlapping_sorted_unroll}$:

assumes $xs \neq []$ **shows** $\text{non_overlapping_sorted } (x \# xs) = (\text{upper } x \leq \text{lower } (\text{hd } xs) \wedge \text{non_overlapping_sorted } xs)$
 $\langle \text{proof} \rangle$

lemma $\text{contiguous_non_overlapping}$: $\langle \text{contiguous } (is::'a::\{\text{preorder}\} \text{ interval list}) \implies \text{non_overlapping_sorted } is \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{cmp_non_adjacent} = (\lambda x y . \text{upper } x < \text{lower } y) \rangle$

lemma $\text{cmp_non_adjacent_lower}$: $\langle \text{cmp_non_adjacent } x y \implies \text{lower } x < \text{lower } y \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{cmp_non_adjacent_upper}$: $\langle \text{cmp_non_adjacent } x y \implies \text{upper } x < \text{upper } y \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{non_adjacent_sorted_wrt_lower} = \text{sorted_wrt } \text{cmp_non_adjacent} \rangle$

lemma $\text{non_adjacent_sorted_wrt_lower_unroll}$:

assumes $xs \neq []$
shows $\text{non_adjacent_sorted_wrt_lower } (x \# xs) =$
 $((\text{upper } x < \text{lower } (\text{hd } xs)) \wedge \text{non_adjacent_sorted_wrt_lower } xs)$
 $\langle \text{proof} \rangle$

lemma $\text{non_adjacent_implies_non_overlapping}$:

assumes $\langle \text{non_adjacent_sorted_wrt_lower } is \rangle$ **shows** $\langle \text{non_overlapping_sorted } is \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{non_overlapping_implies_sorted_wrt_lower}$:

assumes $\langle \text{non_overlapping_sorted } (is::'a::\{\text{minus_mono}\} \text{ interval list}) \rangle$
shows $\langle \text{sorted_wrt_lower } is \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{non_overlapping_implies_sorted_wrt_upper}$:

assumes $\langle \text{non_overlapping_sorted } is \rangle$
shows $\langle \text{sorted_wrt_upper } is \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{non_adjacent_implies_sorted_wrt_lower}$:

assumes $\langle \text{non_adjacent_sorted_wrt_lower } is \rangle$
shows $\langle \text{sorted_wrt_lower } is \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{non_adjacent_implies_distinct}$:

assumes $\langle \text{non_adjacent_sorted_wrt_lower } is \rangle$
shows $\langle \text{distinct } is \rangle$
 $\langle \text{proof} \rangle$

```

fun merge_overlapping_intervals_sorted_wrt_lower :: 'a::linorder interval list  $\Rightarrow$  'a interval list where
merge_overlapping_intervals_sorted_wrt_lower [] = [] |
merge_overlapping_intervals_sorted_wrt_lower [x] = [x] |
merge_overlapping_intervals_sorted_wrt_lower (x#y#ys) =
  ( if upper x  $\leq$  lower y
    then x#(merge_overlapping_intervals_sorted_wrt_lower (y#ys))
    else merge_overlapping_intervals_sorted_wrt_lower ((mk_interval(lower x, max (upper x) (upper y)))#ys)
  )

```

lemma sorted_wrt_lower_unroll:

```

assumes xs  $\neq$  []
shows sorted_wrt_lower (x # xs) = ((if lower x  $\neq$  lower (hd xs)
  then lower x < lower (hd xs)
  else width x  $\leq$  width (hd xs) )  $\wedge$  sorted_wrt_lower (xs))

```

<proof>

lemma sorted_wrt_upper_unroll:

```

assumes xs  $\neq$  []
shows sorted_wrt_upper (x # xs) = ((upper x  $\leq$  upper (hd xs))  $\wedge$  sorted_wrt_upper (xs))

```

<proof>

lemma sorted_wrt_lower_tail: sorted_wrt_lower (x # xs) \implies sorted_wrt_lower (xs)

<proof>

lemma sorted_wrt_lower_tail':sorted_wrt_lower (x # y # ys) \implies sorted_wrt_lower (x # ys)

<proof>

lemma iList_plus_leq_B:

```

assumes sorted_wrt_lower A and sorted_wrt_lower B and 1 < length B
shows hd (map lower (iList_plus A B))  $\leq$  hd (map lower (iList_plus A (tl B)))

```

<proof>

lemma iList_plus_leq_A:

```

assumes sorted_wrt_lower A and sorted_wrt_lower B and 1 < length A
shows hd (map lower (iList_plus A B))  $\leq$  hd (map lower (iList_plus (tl A) B))

```

<proof>

value <merge_overlapping_intervals_sorted_wrt_lower [mk_interval(1::int,2), mk_interval(2,3), mk_interval(5,7), mk_interval(6,10)]>

lemma merge_overlapping_intervals_sorted_wrt_lower_non_nil:

```

assumes <xs  $\neq$  []>
shows <(merge_overlapping_intervals_sorted_wrt_lower xs)  $\neq$  []>

```

<proof>

lemma merge_overlapping_intervals_sorted_hd_lower:

```

assumes <xs  $\neq$  []>
shows lower (hd (merge_overlapping_intervals_sorted_wrt_lower (xs))) = lower (hd xs)

```

<proof>

lemma merge_overlapping_intervals_sorted_hd_upper:

```

assumes <xs  $\neq$  []>

```

shows $\text{upper} (\text{hd } xs) \leq \text{upper} (\text{hd} (\text{merge_overlapping_intervals_sorted_wrt_lower } xs))$
<proof>

lemma *Interval_id[simp]*: $\langle \text{Interval} (\text{lower } x, \text{upper } x) = x \rangle$
<proof>

lemma *mk_interval_id[simp]*: $\langle (\text{mk_interval} (\text{lower } x, \text{upper } x)) = x \rangle$
<proof>

lemma *merge_overlapping_intervals_sorted_hd_width*:

assumes $\langle xs \neq [] \rangle$
shows $\text{width} (\text{hd } xs) \leq \text{width} (\text{hd} (\text{merge_overlapping_intervals_sorted_wrt_lower } (xs::'a::\{\text{minus_mono}\} \text{ interval list})))$
<proof>

lemma *merge_overlapping_intervals_sorted_wrt_lower_sorted_lower*:

assumes $\langle \text{sorted_wrt_lower } (xs::'a::\{\text{minus_mono}\} \text{ interval list}) \rangle$
shows $\langle \text{sorted_wrt_lower} (\text{merge_overlapping_intervals_sorted_wrt_lower } xs) \rangle$
<proof>

lemma *merge_overlapping_intervals_sorted_sorted_non_non_overlapping*:

assumes $\langle \text{sorted_wrt_lower } (xs::'a::\{\text{minus_mono}\} \text{ interval list}) \rangle$
shows $\langle \text{non_overlapping_sorted} (\text{merge_overlapping_intervals_sorted_wrt_lower } xs) \rangle$
<proof>

fun *merge_adjacent_intervals_sorted_wrt_lower* :: $'a::\text{linorder}$ interval list $\Rightarrow 'a$ interval list **where**

merge_adjacent_intervals_sorted_wrt_lower [] = [] |
merge_adjacent_intervals_sorted_wrt_lower [x] = [x] |
merge_adjacent_intervals_sorted_wrt_lower (x#y#ys) =
 (if upper x < lower y
 then x#(merge_adjacent_intervals_sorted_wrt_lower (y#ys))
 else merge_adjacent_intervals_sorted_wrt_lower ((mk_interval(lower x, max (upper y) (upper x)))#ys)
)

value $\langle \text{merge_adjacent_intervals_sorted_wrt_lower} \ [\text{mk_interval}(1::\text{int},2), \ \text{mk_interval}(2,3), \ \text{mk_interval}(5,7), \ \text{mk_interval}(6,10)] \rangle$

lemma *merge_adjacent_intervals_sorted_wrt_lower_non_nil*:

assumes $\langle xs \neq [] \rangle$
shows $\langle (\text{merge_adjacent_intervals_sorted_wrt_lower } xs) \neq [] \rangle$
<proof>

lemma *merge_adjacent_intervals_sorted_wrt_lower_non_nil'*:

shows $\langle (\text{merge_adjacent_intervals_sorted_wrt_lower } (x\#xs)) \neq [] \rangle$
<proof>

lemma *merge_adjacent_intervals_sorted_wrt_lower_sorted_lower_lower_hd*:

assumes $\langle \text{sorted_wrt_lower } xs \rangle$
shows $\langle \text{lower} (\text{hd} (\text{merge_adjacent_intervals_sorted_wrt_lower } xs)) = \text{lower} (\text{hd } xs) \rangle$
<proof>

lemma *merge_adjacent_intervals_sorted_wrt_lower_sorted_lower_lower_subset*:
 $\langle \text{set } (\text{map lower } (\text{merge_adjacent_intervals_sorted_wrt_lower } xs)) \subseteq \text{set } (\text{map lower } xs) \rangle$
<proof>

lemma *merge_adjacent_intervals_sorted_wrt_lower_set_eq*:
assumes $\langle \text{set } (xs :: \text{real interval list}) = \text{set } ys \rangle$
and $\langle \text{distinct } xs \rangle$ **and** $\langle \text{distinct } ys \rangle$
and $\langle \text{sorted_wrt_lower } xs \rangle$ **and** $\langle \text{sorted_wrt_lower } ys \rangle$
shows $\langle \text{merge_adjacent_intervals_sorted_wrt_lower } xs = \text{merge_adjacent_intervals_sorted_wrt_lower } ys \rangle$
<proof>

lemma *merge_adjacent_intervals_sorted_wrt_lower_lower_upper*:
assumes *sorted_wrt_lower xs*
shows $x \in \text{set } (\text{merge_adjacent_intervals_sorted_wrt_lower } xs) \implies \exists l \in \text{set } xs. \exists u \in \text{set } xs. \text{lower } l = \text{lower } x \wedge \text{upper } u = \text{upper } x$
<proof>

primrec *interval_insert_sort_lower_width* :: $('a :: \{ \text{linorder, minus} \}) \text{ interval} \Rightarrow 'a \text{ interval list} \Rightarrow 'a \text{ interval list}$ **where**
interval_insert_sort_lower_width $x \ [] = [x]$ |
interval_insert_sort_lower_width $x (y \# ys) =$
(if cmp_lower_width $x y$ *then* $(x \# y \# ys)$ *else* $y \# (\text{interval_insert_sort_lower_width } x \ ys)$ *)*

lemma *interval_insert_sort_lower_width_length*:
 $\langle \text{length } (\text{interval_insert_sort_lower_width } x \ xs) = 1 + \text{length } xs \rangle$
<proof>

lemma *interval_insert_sort_lower_width_nonempty*:
 $\langle \text{interval_insert_sort_lower_width } x \ xs \neq [] \rangle$
<proof>

lemma *interval_insert_sort_wrt_lower*:
 $\langle \text{sorted_wrt_lower } xs \implies \text{sorted_wrt_lower } (\text{interval_insert_sort_lower_width } x \ xs) \rangle$
<proof>

lemma *interval_isort_elements*: $\text{set } (\text{interval_insert_sort_lower_width } x \ xs) = \{x\} \cup \text{set } xs$
<proof>

lemma *foldr_isort_elements*: $\text{set } (\text{foldr } \text{interval_insert_sort_lower_width } xs \ []) = \text{set } xs$
<proof>

definition *interval_sort_lower_width* :: $('a :: \{ \text{linorder, minus} \}) \text{ interval list} \Rightarrow 'a \text{ interval list}$ **where**
interval_sort_lower_width $xs = \text{foldr } \text{interval_insert_sort_lower_width } xs \ []$

lemma *interval_sort_lower_width_length*: $\langle \text{length } (\text{interval_sort_lower_width } xs) = (\text{length } xs) \rangle$
<proof>

lemma *interval_sort_lower_width_sorted*: $\langle \text{sorted_wrt_lower } (\text{interval_sort_lower_width } xs) \rangle$
<proof>

lemma *interval_sort_lower_width_set_eq*:
 $\langle \text{set } (\text{interval_sort_lower_width } x) = \text{set } x \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_sort_lower_width_remdups*:
 $\langle \text{remdups } (\text{interval_sort_lower_width } (\text{remdups } xs)) = \text{interval_sort_lower_width } (\text{remdups } xs) \rangle$
 $\langle \text{proof} \rangle$

lemma *interval_sort_lower_width_distinct*:
assumes $\langle \text{distinct } xs \rangle$ **shows**
 $\langle \text{distinct } (\text{interval_sort_lower_width } (\text{remdups } xs)) \rangle$
 $\langle \text{proof} \rangle$

lemma *foldr_interval_insert_sort_lower_width_distinct*:
assumes $\langle \text{distinct } zs \rangle$
shows $\langle \text{distinct } (\text{foldr } \text{interval_insert_sort_lower_width } zs \ []) \rangle$
 $\langle \text{proof} \rangle$

lemma *non_overlapping_sorted_remdups*:
 $\text{non_overlapping_sorted } xs \implies \text{non_overlapping_sorted } (\text{remdups } xs)$
 $\langle \text{proof} \rangle$

lemma *insert_in_lower_width*: $x \in \text{set } (\text{interval_insert_sort_lower_width } a \text{ list}) = (x = a \vee x \in \text{set list})$
 $\langle \text{proof} \rangle$

lemma *remdups_set_eq*:
assumes $\langle \text{set } xs = \text{set } ys \rangle$
shows $\langle \text{set } (\text{remdups } xs) = \text{set } (\text{remdups } ys) \rangle$
 $\langle \text{proof} \rangle$

lemma *remdups_lower_hd*:
assumes $xs \neq []$ **and** *sorted_wrt_lower* xs
shows $(\text{lower } \circ \text{hd}) (\text{remdups } xs) = (\text{lower } \circ \text{hd}) xs$
 $\langle \text{proof} \rangle$

8.1.4 Various Notions of Validity of Sorted Lists of Intervals

Validity Tests

definition $\langle \text{valid_mInterval_ovl } is = (\text{sorted_wrt_lower } is \wedge \text{distinct } is \wedge is \neq []) \rangle$

The predicate *valid_mInterval_ovl* requires that a list of intervals is distinct and sorted with respect to the lower bound of each interval.

definition *valid_mInterval_adj* :: $'a::\text{minus_mono}$ *interval list* $\implies \text{bool}$
where $\langle \text{valid_mInterval_adj } is = (\text{non_overlapping_sorted } is \wedge \text{distinct } is \wedge is \neq []) \rangle$

The predicate *valid_mInterval_adj* is strictly stronger than *valid_mInterval_ovl*:

lemma *valid_adj_imp_ovl*: $\langle \text{valid_mInterval_adj } x \implies \text{valid_mInterval_ovl } x \rangle$
 $\langle \text{proof} \rangle$

Informally, `valid_mInterval_ovl` further limits the list of intervals to be non-overlapping. Note that adjacent intervals (i.e., intervals that share the same bounds) are allowed. For example:

lemma `valid_mInterval_adj`: $[Interval(1::int,2), Interval(2,3)]$
 $\langle proof \rangle$

definition `valid_mInterval_non_ovl is` = $(valid_mInterval_ovl\ is \wedge non_adjacent_sorted_wrt_lower\ is)$

Informally, `valid_mInterval_non_ovl` further limits the list of intervals to also forbid adjacent intervals (i.e., intervals that share the same bounds) are allowed. It is strictly stronger than the other two predicates:

lemma `valid_non_ovl_imp_ovl`: $\langle valid_mInterval_non_ovl\ x \implies valid_mInterval_ovl\ x \rangle$
 $\langle proof \rangle$

lemma `valid_non_ovl_imp_adj`: $\langle valid_mInterval_non_ovl\ x \implies valid_mInterval_adj\ x \rangle$
 $\langle proof \rangle$

lemma `valid_mInterval_non_ovl_sorted`: $valid_mInterval_non_ovl\ xs \implies sorted_wrt_lower\ xs$
 $\langle proof \rangle$

lemma `valid_mInterval_non_ovl_unroll`:
 $\langle ys \neq [] \implies valid_mInterval_non_ovl\ (y \# ys) \implies valid_mInterval_non_ovl\ ys \rangle$
 $\langle proof \rangle$

lemma `valid_mInterval_non_ovl_eq1`:
assumes $\langle valid_mInterval_non_ovl\ xs \rangle$
and $\langle valid_mInterval_non_ovl\ ys \rangle$
and $\langle set\ xs = set\ ys \rangle$
shows $\langle xs = ys \rangle$
 $\langle proof \rangle$

Constructors

Overlapping Intervals **definition** `mk_mInterval_ovl` = $remdups\ o\ interval_sort_lower_width$

lemma `mk_mInterval_ovl_non_empty`: $\langle is \neq [] \implies (mk_mInterval_ovl\ is) \neq [] \rangle$
 $\langle proof \rangle$

lemma `mk_mInterval_ovl_empty[simp]`:
 $mk_mInterval_ovl\ [] = []$
 $\langle proof \rangle$

lemma `mk_mInterval_ovl_distinct`: $\langle distinct\ (mk_mInterval_ovl\ is) \rangle$
 $\langle proof \rangle$

lemma `sorted_wrt_lower_remdups`:
 $sorted_wrt_lower\ xs \implies sorted_wrt_lower\ (remdups\ xs)$
 $\langle proof \rangle$

lemma `interval_sort_lower_width_swap_remdups`:
 $\langle remdups\ (interval_sort_lower_width\ xs) = interval_sort_lower_width\ (remdups\ xs) \rangle$
for $xs::'a::\{minus_mono,\ linordered_field\}$ interval list
 $\langle proof \rangle$

lemma *mk_mInterval_ovl_sorted*: $\langle \text{sorted_wrt_lower } (mk_mInterval_ovl\ is) \rangle$
<proof>

theorem *mk_mInterval_ovl_valid*: $\langle is \neq [] \implies \text{valid_mInterval_ovl } (mk_mInterval_ovl\ is) \rangle$
<proof>

lemma *valid_mk_mInterval_ovl_id*:
assumes $\langle \text{valid_mInterval_ovl } xs \rangle$
shows $\langle mk_mInterval_ovl\ xs = xs \rangle$
<proof>

lemma *mk_mInterval_ovl_eq*:
assumes $\langle \text{set } xs = \text{set } (ys::'a::\{\text{minus_mono, linordered_field}\}\ \text{interval list}) \rangle$
shows $\langle mk_mInterval_ovl\ xs = mk_mInterval_ovl\ ys \rangle$
<proof>

Adjacent Intervals **definition** *mk_mInterval_adj* :: $('a::\{\text{minus, linorder, linorder}\} \text{ interval list} \implies 'a \text{ interval list}$
where $\langle mk_mInterval_adj = \text{remdups } o \text{ merge_overlapping_intervals_sorted_wrt_lower } o \text{ mk_mInterval_ovl} \rangle$

lemma *mk_mInterval_adj_non_overlapping_sorted*: $\langle \text{non_overlapping_sorted } (mk_mInterval_adj\ (is::'a::\{\text{minus_mono}\}\ \text{interval list})) \rangle$
<proof>

lemma *mk_mInterval_adj_sorted*: $\langle \text{sorted_wrt_lower } (mk_mInterval_adj\ (is::'a::\{\text{minus_mono}\}\ \text{interval list})) \rangle$
<proof>

lemma *mk_mInterval_adj_non_empty*: $\langle is \neq [] \implies (mk_mInterval_adj\ is) \neq [] \rangle$
<proof>

lemma *mk_mInterval_adj_empty[simp]*:
 $mk_mInterval_adj\ [] = []$
<proof>

lemma *mk_mInterval_adj_distinct*: $\langle \text{distinct } (mk_mInterval_adj\ is) \rangle$
<proof>

theorem *mk_mInterval_adj_valid*: $\langle is \neq [] \implies \text{valid_mInterval_adj } (mk_mInterval_adj\ is) \rangle$
<proof>

lemma *valid_mk_mInterval_adj_id*:
assumes $\langle \text{valid_mInterval_adj } xs \rangle$
shows $\langle mk_mInterval_adj\ xs = xs \rangle$
<proof>

lemma *mk_mInterval_adj_eq*:
assumes $\langle \text{set } xs = \text{set } (ys::'a::\{\text{minus_mono, linordered_field}\}\ \text{interval list}) \rangle$
shows $\langle mk_mInterval_adj\ xs = mk_mInterval_adj\ ys \rangle$
<proof>

lemma *mk_mInterval_ovl_id*:
 $mk_mInterval_ovl\ (mk_mInterval_ovl\ x) = mk_mInterval_ovl\ x$
<proof>

value *valid_mInterval_adj* $(mk_mInterval_adj\ ([lvl\ (1::int)\ 2, lvl\ 1\ 3, lvl\ 1\ 1]))$

value `valid_mInterval_adj (mk_mInterval_adj ([lvl (1::int) 1, lvl 1 2, lvl 2 3]))`

Non-Overlapping Intervals definition `<mk_mInterval_non_ovl = remdups o merge_adjacent_intervals_sorted_wrt_lower o mk_mInterval_ovl>`

lemma `mk_mInterval_non_ovl_distinct:`
`distinct (mk_mInterval_non_ovl is)`
`<proof>`

lemma `mk_mInterval_non_ovl_non_empty:`
`is ≠ [] ⇒ mk_mInterval_non_ovl is ≠ []`
`<proof>`

lemma `mk_mInterval_non_ovl_empty[simp]:`
`mk_mInterval_non_ovl [] = []`
`<proof>`

lemma `mk_mInterval_non_ovl_eq:`
assumes `<set xs = set (ys::'a::{minus_mono, linordered_field} interval list)>`
shows `<mk_mInterval_non_ovl xs = mk_mInterval_non_ovl ys >`
`<proof>`

lemma `sorted_wrt_lower_merge_adjacent_intervals_sorted_wrt_lower:`
`sorted_wrt_lower xs ⇒ sorted_wrt_lower (merge_adjacent_intervals_sorted_wrt_lower xs)`
`<proof>`

lemma `mk_mInterval_non_ovl_sorted_wrt_lower:`
`is ≠ [] ⇒ sorted_wrt_lower (mk_mInterval_non_ovl (is::int interval list))`
`<proof>`

lemma `valid_ovl_mkInterval_non_ovl: is ≠ [] ⇒ valid_mInterval_ovl (mk_mInterval_non_ovl is)`
`<proof>`

lemma `non_adj_sorted_mkInterval_non_ovl:`
`sorted_wrt_lower xs`
`⇒ non_adjacent_sorted_wrt_lower (merge_adjacent_intervals_sorted_wrt_lower xs)`
`<proof>`

lemma `bin_op_mInterval_commute:`
assumes `op_commute: <∧ x y. op x y = op y x>`
shows `<mk_mInterval_non_ovl (bin_op_interval_list op x y) = mk_mInterval_non_ovl (bin_op_interval_list op y x)>`
`<proof>`

lemma `iList_plus_mInterval_ovl_assoc:`
`<mk_mInterval_ovl (iList_plus x (iList_plus y z)) = mk_mInterval_ovl (iList_plus (iList_plus x (y::'a::{minus_mono, linordered_field} interval list)) z)>`
`<proof>`

lemma `iList_plus_mInterval_adj_commute:`
`<mk_mInterval_adj (iList_plus x y) = mk_mInterval_adj (iList_plus y (x::'a::{minus_mono, linordered_field} interval list))>`

list))>
<proof>

lemma *iList_plus_mInterval_non_ovl_assoc*:

<mk_mInterval_non_ovl (iList_plus x (iList_plus y z)) = mk_mInterval_non_ovl (iList_plus (iList_plus x (y::'a::{minus_mono, linordered_field} interval list)) z)>
<proof>

lemma *iList_plus_mInterval_non_ovl_commute*:

<mk_mInterval_non_ovl (iList_plus x y) = mk_mInterval_non_ovl (iList_plus y (x::'a::{minus_mono, linordered_field} interval list))>
<proof>

lemma *iList_plus_mInterval_adj_assoc*:

<mk_mInterval_non_ovl (iList_plus x (iList_plus y z)) = mk_mInterval_non_ovl (iList_plus (iList_plus x (y::'a::{minus_mono, linordered_field} interval list)) z)>
<proof>

lemma *sorted_wrt_lower_mk_mInterval_non_ovl*: sorted_wrt_lower (mk_mInterval_non_ovl xs)

<proof>

theorem *mk_mInterval_non_ovl_valid*: <sorted_wrt_lower is \implies is \neq [] \implies valid_mInterval_non_ovl (mk_mInterval_non_ovl is)>

<proof>

lemma *valid_mk_mInterval_non_ovl_id*:

assumes <valid_mInterval_non_ovl xs>

shows <mk_mInterval_non_ovl xs = xs>

<proof>

lemma *mk_mInterval_non_ovl_single*:

mk_mInterval_non_ovl [x] = [x]

<proof>

lemma *mk_mInterval_non_ovl_id*:

mk_mInterval_non_ovl (mk_mInterval_non_ovl x) = mk_mInterval_non_ovl x

<proof>

value *valid_mInterval_non_ovl* (mk_mInterval_non_ovl ([lvl (1::int) 2, lvl 1 3, lvl 1 1]))

value *valid_mInterval_non_ovl* (mk_mInterval_non_ovl ([lvl (1::int) 1, lvl 1 2, lvl 2 3]))

value<mk_mInterval_ovl [mk_interval ((1::int), 4), mk_interval (0,2), mk_interval (3,5), mk_interval (5,7), mk_interval (7,7), mk_interval (8,8)]>

value<mk_mInterval_adj [mk_interval ((1::int), 4), mk_interval (0,2), mk_interval (3,5), mk_interval (5,7), mk_interval (7,7), mk_interval (8,8)]>

value<mk_mInterval_non_ovl [mk_interval ((1::int), 4), mk_interval (0,2), mk_interval (3,5), mk_interval (5,7), mk_interval (7,7), mk_interval (8,8)]>

8.1.5 Union over a List of Intervals

definition <set_of_interval_list XS = foldr ($\lambda x a. \text{set_of } x \cup a$) XS {}>

lemma *set_of_interval_list_nonempty*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
shows $\langle \text{set_of_interval_list } XS \neq \{\} \rangle$
 $\langle \text{proof} \rangle$

lemma *set_of_interval_list_bdd_below*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
shows $\langle \text{bdd_below } (\text{set_of_interval_list } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *set_of_interval_list_bdd_above*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
shows $\langle \text{bdd_above } (\text{set_of_interval_list } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *inf_set_of_interval_list_lower*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
and *sorted*: $\langle \text{sorted_wrt_lower } XS \rangle$
shows $\langle \text{Inf } (\text{set_of_interval_list } XS) = \text{lower } (\text{hd } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *contiguous_sorted_wrt_upper*:
assumes *contiguous* (*xs*:: *real interval list*)
shows *sorted_wrt_upper xs*
 $\langle \text{proof} \rangle$

lemma *contiguous_sorted_wrt_lower*:
assumes *contiguous* (*XS*:: *real interval list*)
shows *sorted_wrt_lower XS*
 $\langle \text{proof} \rangle$

lemma *max_last_sorted_wrt_upper*:
assumes $\langle XS \neq [] \text{ sorted_wrt_upper } (XS :: 'a :: \{\text{linorder}\} \text{ interval list}) \rangle$
shows $\langle \text{Max } (\text{set } (\text{map } \text{upper } XS)) = \text{upper } (\text{last } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *min_hd_sorted_wrt_lower*:
assumes $\langle XS \neq [] \text{ sorted_wrt_lower } (XS :: 'a :: \{\text{linorder}, \text{minus}, \text{preorder}\} \text{ interval list}) \rangle$
shows $\langle \text{Min } (\text{set } (\text{map } \text{lower } XS)) = \text{lower } (\text{hd } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *lower_isort*:
assumes $\langle xs \neq [] \text{ and } \langle (\text{lower } \circ \text{hd}) \text{ xs} = \text{Min } (\text{lower } ' (\text{set } xs)) \rangle \rangle$
shows $\langle (\text{lower } \circ \text{hd}) (\text{interval_sort_lower_width } xs) = (\text{lower } \circ \text{hd}) \text{ xs} \rangle$
 $\langle \text{proof} \rangle$

lemma *min_sort*:
 $\langle \text{Min } (\text{set } (\text{map } \text{lower } (\text{foldr } \text{interval_insert_sort_lower_width } xs []))) = \text{Min } (\text{set } (\text{map } \text{lower } xs)) \rangle$
 $\langle \text{proof} \rangle$

lemma *mk_mInterval_lower*:

assumes $xs \neq []$
shows $\text{Min} (\text{set} (\text{map lower} (\text{mk_mInterval_non_ovl } xs))) = \text{Min} (\text{set} (\text{map lower } xs))$
<proof>

lemma *sup_set_of_interval_list_upper*:
assumes *non_empty*: $\langle XS \neq ([]::\text{real interval list}) \rangle$
and *sorted*: $\langle \text{sorted_wrt_upper } XS \rangle$
shows $\langle \text{Sup} (\text{set_of_interval_list } XS) = \text{upper} (\text{last } XS) \rangle$
<proof>

lemma *compact_set_of_interval_list*:
 $\langle \text{compact} (\text{set_of_interval_list } (XS::('a::\{\text{preorder, ordered_euclidean_space, topological_space}\} \text{ interval list}))) \rangle$
<proof>

lemma *lower_le_upper_aux*: $\langle xs \neq [] \implies \text{non_overlapping_sorted } xs \implies \text{lower} (\text{hd } xs) \leq \text{upper} (\text{last } xs) \rangle$
<proof>

lemma *contiguous_lower_le_upper*:
assumes *non_empty*: $\langle XS \neq ([]::\text{real interval list}) \rangle$
and *contiguous*: $\langle \text{contiguous } XS \rangle$
shows $\langle (\text{lower} (\text{hd } XS)) \leq (\text{upper} (\text{last } XS)) \rangle$
<proof>

lemma *diameter_Sup_Inf*:
assumes $\langle \text{compact } X \rangle \langle X \neq \{\} \rangle$
shows $\langle \text{diameter } X \leq \text{Sup } X - \text{Inf } X \rangle$
<proof>

lemma *diameter_width_compact*:
assumes $\langle \text{compact } X \rangle \langle \text{bdd_below } X \rangle \langle \text{bdd_above } X \rangle \langle X \neq \{\} \rangle$
shows $\langle \text{diameter } X = \text{Sup } X - \text{Inf } X \rangle$
<proof>

lemma *diameter_contiguous*:
assumes *non_empty*: $\langle XS \neq ([]::\text{real interval list}) \rangle$
and *contiguous*: $\langle \text{contiguous } XS \rangle$
shows $\langle \text{diameter} (\text{set_of_interval_list } XS) = \text{dist} (\text{lower} (\text{hd } XS)) (\text{upper} (\text{last } XS)) \rangle$
<proof>

lemma *interval_list_union_contiguous_lower*:
assumes *non_empty*: $\langle XS \neq [] \rangle$
and *sorted*: $\langle \text{sorted_wrt_lower } XS \rangle$
shows $\langle \text{lower} (\text{interval_list_union } XS) = \text{lower} (\text{hd } XS) \rangle$
<proof>

lemma *interval_list_union_contiguous_upper*:
assumes *non_empty*: $\langle XS \neq [] \rangle$
and *sorted*: $\langle \text{sorted_wrt_upper } XS \rangle$
shows $\langle \text{upper} (\text{interval_list_union } XS) = \text{upper} (\text{last } XS) \rangle$
<proof>

lemma *interval_list_union_contiguous*:
assumes *non_empty*: $\langle XS \neq [] \rangle$
and *sorted_lower*: $\langle \text{sorted_wrt_lower } XS \rangle$
and *sorted_upper*: $\langle \text{sorted_wrt_upper } XS \rangle$
shows $\langle \text{interval_list_union } XS = \text{Interval } (\text{lower } (\text{hd } XS), \text{upper } (\text{last } XS)) \rangle$
 $\langle \text{proof} \rangle$

lemma *contiguous_bounds_lower*:
assumes *non_empty*: $\langle XS \neq [] \rangle$
and *contiguous*: $\langle \text{contiguous } (XS::\text{real interval list}) \rangle$
shows $\text{lower } (\text{hd } XS) = \text{Min } (\text{set } (\text{map } \text{lower } XS))$
 $\langle \text{proof} \rangle$

lemma *contiguous_bounds_upper*:
assumes *non_empty*: $\langle XS \neq [] \rangle$
and *contiguous*: $\langle \text{contiguous } (XS::\text{real interval list}) \rangle$
shows $\text{upper } (\text{last } XS) = \text{Max } (\text{set } (\text{map } \text{upper } XS))$
 $\langle \text{proof} \rangle$

lemma *set_of_interval_list_contiguous*:
assumes *non_empty*: $\langle XS \neq ([]::\text{real interval list}) \rangle$
and *contiguous*: $\langle \text{contiguous } XS \rangle$
shows $\langle \text{set_of_interval_list } XS = \{ \text{lower } (\text{hd } XS) .. \text{upper } (\text{last } XS) \} \rangle$
 $\langle \text{proof} \rangle$

lemma *set_of_interval_list_set_eq_interval_list_union_contiguous*:
assumes *non_empty*: $\langle XS \neq ([]::\text{real interval list}) \rangle$
and *contiguous*: $\langle \text{contiguous } XS \rangle$
shows $\langle \text{set_of_interval_list } XS = \text{set_of } (\text{interval_list_union } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_ovl_lower_hd_min*:
 $\langle \text{valid_mInterval_ovl } x \implies \text{Min } (\text{set } (\text{map } \text{lower } x)) = (\text{lower } \circ \text{hd}) x \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_adj_lower_hd_min*:
 $\langle \text{valid_mInterval_adj } x \implies \text{Min } (\text{set } (\text{map } \text{lower } x)) = (\text{lower } \circ \text{hd}) x \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_non_ovl_lower_hd_min*:
 $\langle \text{valid_mInterval_non_ovl } x \implies \text{Min } (\text{set } (\text{map } \text{lower } x)) = (\text{lower } \circ \text{hd}) x \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_ovl_lower_last_max*:
 $\langle \text{valid_mInterval_ovl } x \implies (\text{Max } (\text{set } (\text{map } \text{lower } x))) = (\text{lower } \circ \text{last}) x \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_adj_upper_hd_min*:
 $\langle \text{valid_mInterval_adj } x \implies \text{Min } (\text{set } (\text{map } \text{upper } x)) = (\text{upper } \circ \text{hd}) x \rangle$
 $\langle \text{proof} \rangle$

lemma *mInterval_adj_upper_last_max*:

⟨ $\text{valid_mInterval_adj } x \implies \text{Max } (\text{set } (\text{map } \text{upper } x)) = (\text{upper } \circ \text{last}) x$ ⟩
⟨ *proof* ⟩

lemma *set_of_subeq_aux*:

⟨ $(\bigcup x \in \text{set } \text{is. } \{\text{lower } x .. \text{upper } x\}) \subseteq \{\text{Min } (\text{lower } ' (\text{set } \text{is})) .. \text{Max } (\text{upper } ' (\text{set } \text{is}))\}$ ⟩
⟨ *proof* ⟩

lemma *lower_merge_adjacent_intervals*:

assumes $x s \neq []$

and $\langle \text{sorted_wrt_lower } x s \rangle$

shows $(\text{lower } \circ \text{hd}) (\text{merge_adjacent_intervals_sorted_wrt_lower } x s) = (\text{lower } \circ \text{hd}) x s$

⟨ *proof* ⟩

lemma *sorted_wrt_lower_hd_min*:

⟨ $x \neq [] \implies \text{sorted_wrt_lower } x \implies \text{Min } (\text{set } (\text{map } \text{lower } x)) = (\text{lower } \circ \text{hd}) x$ ⟩

⟨ *proof* ⟩

lemma *lower_hd_min_over_mk_mInterval_non_ovl*:

$x s \neq [] \implies (\text{lower } \circ \text{hd}) x s = \text{Min } (\text{lower } ' (\text{set } x s)) \implies (\text{lower } \circ \text{hd}) (\text{mk_mInterval_non_ovl } x s) = (\text{lower } \circ \text{hd}) x s$

⟨ *proof* ⟩

theorem *valid_mInterval_non_ovl_nonempty*: $\text{valid_mInterval_non_ovl } x \implies x \neq []$

⟨ *proof* ⟩

end

9 Subdivisions and Refinements

(Lipschitz_Subdivisions_Refinements)

theory

Lipschitz_Subdivisions_Refinements

imports

Lipschitz_Interval_Extension

Multi_Interval_Preliminaries

begin

9.1 Subdivisions

A uniform subdivision of an interval X splits X into a vector of equal length, contiguous intervals.

definition *uniform_subdivision* :: 'a::linordered_field interval \Rightarrow nat \Rightarrow 'a interval list **where**
uniform_subdivision A n = map (λi . let $i' = \text{of_nat } i$ in
 mk_interval (lower A + (upper A - lower A) * $i' / \text{of_nat } n$,
 lower A + (upper A - lower A) * ($i' + 1$) / $\text{of_nat } n$)) [0.. n]

The definition *uniform_subdivision* refers to definition 6.2 in [4]

definition *overlapping_ordered* :: 'a::{preorder} interval list \Rightarrow bool **where**
overlapping_ordered xs = ($\forall i$. $i < \text{length } xs - 1 \longrightarrow \text{lower } (xs ! (i + 1)) \leq \text{upper } (xs ! i)$)

definition *overlapping_non_zero_width* :: 'a::{preorder, minus, zero, ord} interval list \Rightarrow bool **where**
overlapping_non_zero_width xs = ($\forall i < \text{length } xs - 1$. $\exists e$. $e \in_i (xs ! (i + 1)) \wedge e \in_i (xs ! i) \wedge 0 < \text{width } (xs ! (i + 1)) \wedge 0 < \text{width } (xs ! i)$)

definition *overlapping* :: 'a::{preorder} interval list \Rightarrow bool **where**
overlapping xs = ($\forall i < \text{length } xs - 1$. $\exists e$. $e \in_i (xs ! (i + 1)) \wedge e \in_i (xs ! i)$)

definition *check_is_uniform_subdivision* :: 'a::linordered_field interval \Rightarrow 'a interval list \Rightarrow bool **where**
check_is_uniform_subdivision A xs = (let $n = \text{length } xs$ in
 if $n = 0$ then True
 else
 let $d = \text{width } A / \text{of_nat } n$ in
 list_all (λx . $\text{width } x = d$) xs \wedge
 contiguous xs \wedge
 lower (hd xs) = lower A \wedge
 upper (last xs) = upper A)

lemma *non_empty_subdivision*:

assumes $0 < n$

shows *uniform_subdivision* A n $\neq []$

<proof>

lemma *uniform_subdivision_id*: *uniform_subdivision* X 1 = [X]

<proof>

lemma *subdivision_length_n*:

assumes $0 < n$

shows $\text{length}(\text{uniform_subdivision } A \ n) = n$

<proof>

lemma *contiguous_uniform_subdivision*: *contiguous* (*uniform_subdivision* *A* *n*)

<proof>

lemma *overlapping_ordered_uniform_subdivision*:

assumes $0 < n$

shows *overlapping_ordered* (*uniform_subdivision* *A* *n*)

<proof>

lemma *overlapping_uniform_subdivision*:

assumes $0 < N$

shows *overlapping* (*uniform_subdivision* *X* *N*)

<proof>

lemma *hd_lower_uniform_subdivision*:

assumes $0 < n$

shows $\text{lower} (\text{hd} (\text{uniform_subdivision } A \ n)) = \text{lower } A$

<proof>

lemma *last_upper_uniform_subdivision*:

assumes $0 < n$

shows $\text{upper} (\text{last} (\text{uniform_subdivision } A \ n)) = \text{upper } A$

<proof>

lemma *uniform_subdivisions_width_single*:

fixes *A* :: 'a::linordered_field interval

shows $\langle \text{width} (\text{Interval} (\text{lower } A + (\text{upper } A - \text{lower } A) * x / \text{of_nat } n, \text{lower } A + (\text{upper } A - \text{lower } A) * (x + 1) / \text{of_nat } n)) = \text{width } A / \text{of_nat } n \rangle$

<proof>

lemma *uniform_subdivisions_width*:

assumes $0 < n$

shows $\langle \forall A. A \in \text{set} (\text{uniform_subdivision } X \ n) \longrightarrow \text{width } A = \text{width } X / \text{of_nat } n \rangle$

<proof>

lemma *uniform_subdivision_sum_width*:

assumes $0 < n$

shows $\langle \text{sum_list} (\text{map } \text{width} (\text{uniform_subdivision } X \ n)) = \text{width } X \rangle$

<proof>

lemma *uniform_subdivisions_distinct*:

assumes $0 < n$ $0 < \text{width } A$

shows *distinct* (*uniform_subdivision* *A* *n*)

<proof>

lemma *uniform_subdivisions_non_overlapping*:

assumes $0 < n$

shows *non_overlapping_sorted* (*uniform_subdivision* *A* *n*)

⟨proof⟩

We prove that our uniform subdivision meets the multi-interval type

lemma *uniform_subdivisions_valid_ainterval*:
assumes $0 < n$ $0 < \text{width } A$
shows *valid_mInterval_adj*(*uniform_subdivision* A n)
⟨proof⟩

lemma *uniform_subdivisions_valid*:
assumes $0 < n$
shows *check_is_uniform_subdivision* A (*uniform_subdivision* A n)
⟨proof⟩

9.2 Refinement

Let $F X$ be an inclusion isotonic, Lipschitz, interval extension for $X \subseteq Y$. A refinement $F_N X$ of $F X$ is the union of interval values of X over the elements of a uniform subdivision of X

definition *refinement* :: ($'a::\{\text{linordered_field}, \text{lattice}\}$ interval $\Rightarrow 'a$ interval) \Rightarrow nat $\Rightarrow 'a$ interval $\Rightarrow 'a$ interval **where**
⟨*refinement* $F N X = (\text{interval_list_union } (\text{map } F (\text{uniform_subdivision } X N)))$ ⟩

definition *check_is_refinement where*
⟨*check_is_refinement* $F n A s B = (\text{let } I = \text{refinement } F n A s \text{ in } \text{lower } B \leq \text{lower } I \wedge \text{upper } I \leq \text{upper } B)$ ⟩

definition *refinement_set* :: ($'a::\{\text{linordered_field}, \text{lattice}\}$ interval $\Rightarrow 'a$ interval) \Rightarrow nat $\Rightarrow 'a$ interval $\Rightarrow 'a$ set **where**
⟨*refinement_set* $F N X = (\text{set_of_interval_list } (\text{map } F (\text{uniform_subdivision } X N)))$ ⟩

The definition *refinement* refers to definition 6.3 in [4].

The excess width of $F X$ is $w(E X) = w(F X) - w(f X)$. The united extension $f x$ for $x \in X$ has zero excess width and we can compute $f x$ as closely as desired by computing refinements of an extension $F X$.

definition *width_set* $s = \text{Sup } s - \text{Inf } s$

lemma *width_set_bounded*:
fixes $X :: \langle \text{real set} \rangle$
assumes $\langle \text{bdd_below } X \rangle$ $\langle \text{bdd_above } X \rangle$
shows $\langle \forall x \in X. \forall x' \in X. \text{dist } x x' \leq \text{width_set } X \rangle$
⟨proof⟩

lemma *width_inclusion_isotonic_approx*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes *inclusion_isotonic* F *is_interval_extension_of* f
shows $\langle 0 \leq \text{width } (F X) - \text{width_set } (f' \text{ set_of } X) \rangle$
⟨proof⟩

lemma *diameter_width*:
assumes $\langle a \leq b \rangle$
shows $\langle \text{diameter } \{a..b\} = \text{width_set } \{a..b\} \rangle$
⟨proof⟩

lemma *lipschitz_dist_diameter_limit*:
fixes $S :: \langle 'a::\{\text{metric_space}, \text{heine_borel}\} \text{ set} \rangle$
and $f :: \langle 'a::\{\text{metric_space}, \text{heine_borel}\} \Rightarrow 'b::\{\text{metric_space}, \text{heine_borel}\} \rangle$
assumes $\langle C - \text{lipschitz_on } S f \rangle$ and $\langle \text{bounded } S \rangle$

shows $\langle x \in (f'S) \implies y \in (f'S) \implies \text{dist } x y \leq \text{diameter } (f'S) \rangle$
 $\langle \text{proof} \rangle$

definition $\text{excess_width_diameter} :: ('a::\text{preorder_interval} \Rightarrow \text{real_interval}) \Rightarrow ('a \Rightarrow 'b::\text{metric_space}) \Rightarrow 'a \text{ interval} \Rightarrow \text{real where}$
 $\langle \text{excess_width_diameter } F f X = \text{width}(F X) - \text{diameter } (f' \text{ set_of } X) \rangle$

definition $\text{excess_width_set} :: ('a::\{\text{minus,linorder,Inf,Sup}\} \text{ interval} \Rightarrow 'a \text{ set}) \Rightarrow ('a \Rightarrow 'a) \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ where}$
 $\langle \text{excess_width_set } F f X = \text{width_set}(F X) - \text{width_set } (f' \text{ set_of } X) \rangle$

definition $\text{excess_width} :: ('a::\{\text{minus,linorder,Inf,Sup}\} \text{ interval} \Rightarrow 'a \text{ interval}) \Rightarrow ('a \Rightarrow 'a) \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ where}$
 $\langle \text{excess_width } F f X = \text{width}(F X) - \text{width_set } (f' \text{ set_of } X) \rangle$

The definition `excess_width` refers to definition 6.4 in [4]

lemma $\text{width_set_of: fixes } X :: \text{real_interval}$
shows $\text{width_set_upper_lower: } \langle \text{width_set } (\text{set_of } X) = |(\text{lower } X) - (\text{upper } X)| \rangle$
 $\langle \text{proof} \rangle$

lemma width_set_dist:
fixes $f :: \text{real} \Rightarrow \text{real}$
shows $\text{width_set } (\text{set_of } X) = (\text{dist } (\text{lower } X) (\text{upper } X))$
 $\langle \text{proof} \rangle$

lemma $\text{diameter_of: fixes } X :: \text{real_interval}$
shows $\text{diameter_upper_lower: } \langle \text{diameter } (\text{set_of } X) = |(\text{lower } X) - (\text{upper } X)| \rangle$
 $\langle \text{proof} \rangle$

lemma diameter_dist:
fixes $X :: \text{real_interval}$
shows $\text{diameter } (\text{set_of } X) = (\text{dist } (\text{lower } X) (\text{upper } X))$
 $\langle \text{proof} \rangle$

lemma $\text{bdd_below_f_set_of:}$
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $C\text{-lipschitz_on } X f$
and $\langle \text{bounded } X \rangle$ **and** $\langle X \neq \{\} \rangle$
shows $\langle \text{bdd_below } (f' X) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{bdd_above_f_set_of:}$
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $C\text{-lipschitz_on } (X) f$
and $\langle \text{bounded } X \rangle$ **and** $\langle X \neq \{\} \rangle$
shows $\langle \text{bdd_above } (f' X) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{diameter_image_dist:}$
fixes $f::\langle \text{real} \Rightarrow \text{real} \rangle$
assumes $\langle \text{continuous_on } (\text{set_of } X) f \rangle$
shows $\langle (\exists x \in \text{set_of } X. \exists x' \in \text{set_of } X. \text{diameter } (f' \text{ set_of } X) = \text{dist } (f x) (f x')) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{excess_width_inf_diameter:}$
fixes $F::\langle \text{real_interval} \Rightarrow \text{real_interval} \rangle$

assumes $\text{inclusion_isotonic } F \text{ } F \text{ is_interval_extension_of } f \text{ } \langle C\text{-lipschitz_on } (\text{set_of } X) \text{ } f \rangle$
shows $\langle \text{dist } (\text{Inf } (f' \text{ set_of } X)) \text{ } (\text{lower } (F X)) \leq \text{excess_width_diameter } F f X \rangle$
 $\langle \text{proof} \rangle$

lemma excess_width_inf :

fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inclusion_isotonic } F \text{ } F \text{ is_interval_extension_of } f \text{ } \langle C\text{-lipschitz_on } (\text{set_of } X) \text{ } f \rangle$
shows $\langle \text{dist } (\text{Inf } (f' \text{ set_of } X)) \text{ } (\text{lower } (F X)) \leq \text{excess_width } F f X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{excess_width_sup_diameter}$:

fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inclusion_isotonic } F \text{ } F \text{ is_interval_extension_of } f \text{ } \langle C\text{-lipschitz_on } (\text{set_of } X) \text{ } f \rangle$
shows $\langle \text{dist } (\text{Sup } (f' \text{ set_of } X)) \text{ } (\text{upper } (F X)) \leq \text{excess_width } F f X \rangle$
 $\langle \text{proof} \rangle$

lemma excess_width_sup :

fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inclusion_isotonic } F \text{ } F \text{ is_interval_extension_of } f \text{ } \langle C\text{-lipschitz_on } (\text{set_of } X) \text{ } f \rangle$
shows $\langle \text{dist } (\text{Sup } (f' \text{ set_of } X)) \text{ } (\text{upper } (F X)) \leq \text{excess_width } F f X \rangle$
 $\langle \text{proof} \rangle$

If $F X$ is an inclusion isotonic, Lipschitz, interval extension then the excess width of a refinement is of order $(1 :: 'a) / N$

If X and X are intervals such that $X \subseteq Y$, then there is an interval E with $\text{lower } E \leq (o :: 'a) \wedge (o :: 'a) \leq \text{upper } E$ such that $Y = X + E$ and $w Y = w X + w E$.

lemma $\text{interval_subset_width}$:

fixes $X Y :: 'a :: \{\text{linordered_ring, lattice}\} \text{ interval}$
assumes $X \leq Y$
and $X_def: X = \text{Interval}(a, b)$ **and** $x_valid: a \leq b$
and $Y_def: Y = \text{Interval}(c, d)$ **and** $y_valid: c \leq d$
shows $\exists E. o \in_i E \wedge Y = X + E \wedge \text{width } Y = \text{width } X + \text{width } E$
 $\langle \text{proof} \rangle$

lemma excess_width_incl :

fixes $F :: \text{real interval} \Rightarrow \text{real interval}$ **and** $X :: \text{real interval}$
assumes $\text{int}: \langle F \text{ is_interval_extension_of } f \rangle$
and $\text{iso}: \text{inclusion_isotonic } F$
and $L\text{-lipschitz_on } (\text{set_of } X) \text{ } f$
shows $\langle \exists E. F X = \text{Interval}(\text{Inf } (f' \text{ set_of } X), \text{Sup } (f' \text{ set_of } X)) + E \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{excess_interval_superset_interval}$:

fixes $F :: \text{real interval} \Rightarrow \text{real interval}$ **and** $X :: \text{real interval}$
assumes $\text{int}: \langle F \text{ is_interval_extension_of } f \rangle$
and $\text{iso}: \text{inclusion_isotonic } F$
and $L\text{-lipschitz_on } (\text{set_of } X) \text{ } f$
and $\text{ex}: \langle \exists E. F X = \text{Interval}(\text{Inf } (f' \text{ set_of } X), \text{Sup } (f' \text{ set_of } X)) + E \rangle$
shows $\langle \text{Interval}(\text{Inf } (f' \text{ set_of } X), \text{Sup } (f' \text{ set_of } X)) \leq F X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{each_subdivision_width_order}$:

fixes $X :: 'a::\{\text{linordered_field}, \text{lattice}, \text{metric_space}\}$ interval
assumes $\text{inclusion_isotonic } F \text{ lipschitz_on } C \ U \ F \ F \text{ is_interval_extension_of } f$
and set $(\text{uniform_subdivision } X \ N) \subseteq U \ o < N \ Xs \in \text{set } (\text{uniform_subdivision } X \ N)$
shows $\text{width}(F \ Xs) \leq C * \text{width}(X) / \text{of_nat } N$
 $\langle \text{proof} \rangle$

lemma $\text{each_subdivision_excess_width_order}$:

fixes $X :: \text{real interval}$
assumes $\text{inclusion_isotonic } F \text{ lipschitz_on } C \ U \ F \ F \text{ is_interval_extension_of } f$
and set $(\text{uniform_subdivision } X \ N) \subseteq U \ o < N$
and $L\text{-lipschitz_on } (\text{set_of } (\text{interval_list_union } (\text{uniform_subdivision } X \ N))) \ f$
shows $\forall Xs \in \text{set } (\text{uniform_subdivision } X \ N) . \text{excess_width } F \ f \ Xs \leq C * \text{width}(X) / \text{of_nat } N$
 $\langle \text{proof} \rangle$

The theorem $\llbracket \text{inclusion_isotonic } ?F; ?C\text{-lipschitz_on } ?U \ ?F; ?F \text{ is_interval_extension_of } ?f; \text{set } (\text{uniform_subdivision } ?X \ ?N) \subseteq ?U; o < ?N; ?Xs \in \text{set } (\text{uniform_subdivision } ?X \ ?N) \rrbracket \implies \text{width } (?F \ ?Xs) \leq ?C * \text{width } ?X / \text{of_nat } ?N$ refers to Theorem 6.1 in [4].

lemma sup_interval_max :

fixes $X \ Y :: 'a::\{\text{linordered_ring}, \text{lattice}\}$ interval
shows $\text{sup } X \ Y = \text{Interval}(\text{min } (\text{lower } X) (\text{lower } Y), \text{max } (\text{upper } X) (\text{upper } Y))$
 $\langle \text{proof} \rangle$

lemma $\text{interval_inf_sup_lower}$: $\text{inf } (\text{lower } l1) (\text{lower } l2) = \text{lower } (\text{sup } l1 \ l2)$

$\langle \text{proof} \rangle$

lemma $\text{interval_sup_sup_upper}$: $\text{sup } (\text{upper } l1) (\text{upper } l2) = \text{upper } (\text{sup } l1 \ l2)$

$\langle \text{proof} \rangle$

lemma $\text{interval_union_lower}$:

assumes $\text{contiguous } Xs \ Xs \neq []$
shows $\text{lower } (\text{interval_list_union } Xs) = \text{lower } (Xs!o)$
 $\langle \text{proof} \rangle$

lemma $\text{interval_union_upper}$:

assumes $\text{contiguous } Xs \ Xs \neq []$
shows $\text{upper } (\text{interval_list_union } Xs) = \text{upper } (\text{last } Xs)$
 $\langle \text{proof} \rangle$

lemma union_set :

assumes $o < n$
shows $\text{interval_list_union } (\text{uniform_subdivision } X \ n) = X$
 $\langle \text{proof} \rangle$

lemma sum_list_less :

assumes $\text{list_all } (\lambda n. n \leq (y::\text{real})) \ xs$
shows $\text{sum_list } xs \leq y * \text{length } xs$
 $\langle \text{proof} \rangle$

lemma in_bounds2 :

fixes $X \ Y :: 'a::\{\text{linordered_ring}\}$ interval
shows $x \in_i X \wedge x \in_i Y \implies$
 $(\text{lower } Y \leq \text{lower } X \wedge \text{upper } Y \leq \text{upper } X \wedge \text{lower } X \leq \text{upper } Y \wedge \text{lower } Y \leq \text{upper } X) \vee$
 $(\text{lower } X \leq \text{lower } Y \wedge \text{upper } X \leq \text{upper } Y \wedge \text{lower } X \leq \text{upper } Y \wedge \text{lower } Y \leq \text{upper } X) \vee$
 $(\text{lower } Y \leq \text{lower } X \wedge \text{upper } X \leq \text{upper } Y \wedge \text{lower } Y \leq \text{lower } X \wedge \text{lower } Y \leq \text{upper } X) \vee$

$(\text{lower } X \leq \text{lower } Y \wedge \text{upper } Y \leq \text{upper } X \wedge \text{lower } X \leq \text{upper } Y \wedge \text{lower } Y \leq \text{upper } X)$
<proof>

lemma overlapping_width_sum:

fixes $X Y :: 'a::\{\text{linordered_ring, lattice}\}$ interval

assumes overlapping $[X, Y]$

shows $\text{width } (\text{sup } X Y) \leq \text{width } X + \text{width } Y$

<proof>

lemma interval_list_union_width:

fixes $xs :: 'a::\{\text{linordered_ring, lattice}\}$ interval list

assumes overlapping xs $xs \neq []$

shows overlapping $xs \implies \text{width } (\text{interval_list_union } xs) \leq \text{sum_list } (\text{map } \text{width } xs)$

<proof>

lemma map_non_zero_width:

fixes $U :: 'a::\{\text{linordered_idom}\}$ interval set

assumes $C\text{-lipschitz_on } U$ F inclusion_isotonic F set $xs \subseteq U$

shows $\forall x \in \text{set } xs. 0 \leq \text{width } x \longrightarrow 0 \leq \text{width } (F x)$

<proof>

lemma inclusion_isotonic_preserves_overlapping:

assumes inclusion_isotonic F $xs \neq []$ F is_interval_extension_of f

shows contiguous $xs \implies \text{overlapping } (\text{map } F xs)$

<proof>

lemma bounded_image_of:

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

assumes $\langle L\text{-lipschitz_on } (\text{set_of } X) f \rangle$

shows $\langle \text{bounded } (f' \text{ set_of } X) \rangle$

<proof>

lemma dist_le_diameter:

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

assumes $\langle C\text{-lipschitz_on } (\text{set_of } X) f \rangle$

shows $\text{dist } (f (\text{upper } X)) (f (\text{lower } X)) \leq \text{diameter } (f' \text{ set_of } X)$

<proof>

lemma excess_width_inf_sup:

fixes $X :: \text{real interval}$ and $f::\langle \text{real} \Rightarrow \text{real} \rangle$

assumes $\langle \text{continuous_on } (\text{set_of } X) f \rangle$

shows $\text{Inf } (f' \text{ set_of } X) - \text{lower } (F X) + \text{upper } (F X) - \text{Sup } (f' \text{ set_of } X) \leq \text{excess_width } F f X$

<proof>

lemma excess_width_lower_bound:

fixes $X :: \text{real interval}$

assumes inclusion_isotonic F F is_interval_extension_of f $\langle \text{continuous_on } (\text{set_of } X) f \rangle$

shows $\text{Inf } (f' \text{ set_of } X) - \text{lower } (F X) \leq \text{excess_width } F f X$

<proof>

lemma excess_width_upper_bound:

fixes $X :: \text{real interval}$

assumes inclusion_isotonic F F is_interval_extension_of f $\langle \text{continuous_on } (\text{set_of } X) f \rangle$

shows $\text{upper}(FX) - \text{Sup}(f' \text{ set_of } X) \leq \text{excess_width } F f X$
 ⟨proof⟩

lemma *lipschitz_excess_width_lower_bound*:
fixes $X :: \text{real interval}$
assumes *inclusion_isotonic* F *lipschitz* L *is_interval_extension_of* f
and $\text{set}(\text{uniform_subdivision } X N) \subseteq U \ N = 1$
and L -*lipschitz* $\text{on}(\text{set_of}(\text{interval_list_union}(\text{uniform_subdivision } X N)))$ f
shows $\text{Inf}(f' \text{ set_of } X) - \text{lower}(FX) \leq C * \text{width } X$
 ⟨proof⟩

lemma *lipschitz_excess_width_upper_bound*:
fixes $X :: \text{real interval}$
assumes *inclusion_isotonic* F *lipschitz* L *is_interval_extension_of* f
and $\text{set}(\text{uniform_subdivision } X N) \subseteq U \ N = 1$
and L -*lipschitz* $\text{on}(\text{set_of}(\text{interval_list_union}(\text{uniform_subdivision } X N)))$ f
shows $\text{upper}(FX) - \text{Sup}(f' \text{ set_of } X) \leq C * \text{width } X$
 ⟨proof⟩

lemma *excess_width_bound_inf*:
fixes $X :: \text{real interval}$
assumes *excess_width_bound*: $\langle \text{excess_width } F f X \leq k \rangle$
and *inclusion_isotonic*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_extension*: $\langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \text{Inf}(f' \text{ set_of } X) - k \leq \text{lower}(FX) \rangle$
 ⟨proof⟩

lemma *excess_width_bound_sup*:
fixes $X :: \text{real interval}$
assumes *excess_width_bound*: $\langle \text{excess_width } F f X \leq k \rangle$
and *inclusion_isotonic*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_extension*: $\langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \text{upper}(FX) \leq \text{Sup}(f' \text{ set_of } X) + k \rangle$
 ⟨proof⟩

lemma *set_of_interval_list_subset_inf_sup*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
shows $\langle \text{set_of_interval_list } XS \subseteq \{ \text{Min}(\text{set}(\text{map } \text{lower } XS)) .. \text{Max}(\text{set}(\text{map } \text{upper } XS)) \} \rangle$
 ⟨proof⟩

lemma *lower_bound_F_inf*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
and *inclusion_isotonic*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_extension*: $\langle F \text{ is_interval_extension_of } f \rangle$
and *sorted_lower*: $\langle \text{sorted_wrt_lower } XS \rangle$
and *lipschitz*: $\langle 0 \leq C \rangle \langle C$ -*lipschitz* $\text{on}(\text{set_of_interval_list } XS)$ f
and *excess_width_bounded*: $\langle \text{Max}(\text{set}(\text{map}(\text{excess_width } F f) XS)) \leq k \rangle$
shows $\langle \text{Inf}(f'(\text{set_of_interval_list } XS)) - k \leq \text{Inf}(\text{set_of_interval_list}(\text{map } F XS)) \rangle$
 ⟨proof⟩

lemma *upper_bound_F_sup*:
assumes *non_empty*: $\langle XS \neq [] :: \text{real interval list} \rangle$
and *inclusion_isotonic*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_extension*: $\langle F \text{ is_interval_extension_of } f \rangle$

and sorted_upper: $\langle \text{sorted_wrt_upper } XS \rangle$
and lipschitz: $\langle o \leq C \rangle \langle C - \text{lipschitz_on } ((\text{set_of_interval_list } XS)) f \rangle$
and excess_width_bounded: $\langle \text{Max } (\text{set } ((\text{map } (\text{excess_width } F f)) XS)) \rangle \leq k \rangle$
shows $\langle \text{Sup } (\text{set_of_interval_list } (\text{map } F XS)) \leq (\text{Sup } (f' (\text{set_of_interval_list } XS))) + k \rangle$
 $\langle \text{proof} \rangle$

lemma Inf_interval_list_approx: **assumes** non_empty: $\langle XS \neq ([] :: \text{real interval list}) \rangle$
and inclusion_isotonic: $\langle \text{inclusion_isotonic } F \rangle$
and interval_extension: $\langle F \text{ is_interval_extension_of } f \rangle$
and sorted_upper: $\langle \text{sorted_wrt_upper } XS \rangle$
and lipschitz: $\langle o \leq C \rangle \langle C - \text{lipschitz_on } ((\text{set_of_interval_list } XS)) f \rangle$
and excess_width_bounded: $\langle \text{Max } (\text{set } ((\text{map } (\text{excess_width } F f)) XS)) \rangle \leq k \rangle$
shows $\langle \text{Inf } (\text{set_of_interval_list } (\text{map } F XS)) \leq \text{Inf } (f' \text{ set_of_interval_list } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma Sup_interval_list_approx: **assumes** non_empty: $\langle XS \neq ([] :: \text{real interval list}) \rangle$
and inclusion_isotonic: $\langle \text{inclusion_isotonic } F \rangle$
and interval_extension: $\langle F \text{ is_interval_extension_of } f \rangle$
and sorted_lower: $\langle \text{sorted_wrt_lower } XS \rangle$
and lipschitz: $\langle o \leq C \rangle \langle C - \text{lipschitz_on } ((\text{set_of_interval_list } XS)) f \rangle$
and excess_width_bounded: $\langle \text{Max } (\text{set } ((\text{map } (\text{excess_width } F f)) XS)) \rangle \leq k \rangle$
shows $\langle \text{Sup } (f' \text{ set_of_interval_list } XS) \leq \text{Sup } (\text{set_of_interval_list } (\text{map } F XS)) \rangle$
 $\langle \text{proof} \rangle$

lemma map_inclusion_isotonic_excess_width_bounded:
assumes non_empty: $\langle XS \neq ([] :: \text{real interval list}) \rangle$
and inclusion_isotonic: $\langle \text{inclusion_isotonic } F \rangle$
and interval_extension: $\langle F \text{ is_interval_extension_of } f \rangle$
and sorted_lower: $\langle \text{sorted_wrt_lower } XS \rangle$
and sorted_upper: $\langle \text{sorted_wrt_upper } XS \rangle$
and lipschitz: $\langle C - \text{lipschitz_on } ((\text{set_of_interval_list } XS)) f \rangle$
and excess_width_bounded: $\langle \text{Max } (\text{set } ((\text{map } (\text{excess_width } F f)) XS)) \rangle \leq k \rangle$
shows $\langle \text{width_set } (\text{set_of_interval_list } (\text{map } F XS)) - \text{width_set } (f' (\text{set_of_interval_list } XS)) \rangle \leq 2 * k \rangle$
and $\langle \text{width_set } (\text{set_of_interval_list } (\text{map } F XS)) - \text{width_set } (f' (\text{set_of_interval_list } XS)) \rangle \geq o \rangle$
 $\langle \text{proof} \rangle$

lemma max_subdivision_excess_width_order:
fixes $X :: \text{real interval}$
assumes inclusion_isotonic F lipschitzl_on $C U F F$ is_interval_extension_of f
and set $(\text{uniform_subdivision } X N) \subseteq U$ $o < N$
and $L - \text{lipschitz_on } (\text{set_of_interval_list } (\text{uniform_subdivision } X N)) f$
shows $\langle \text{Max } (\text{set } (\text{map } (\text{excess_width } F f) (\text{uniform_subdivision } X N))) \rangle \leq C * \text{width } X / \text{real } N \rangle$
 $\langle \text{proof} \rangle$

lemma set_of_interval_list_set_eq_interval_list_union_contiguous:
assumes non_empty: $\langle XS \neq ([] :: \text{real interval list}) \rangle$
and contiguous: $\langle \text{contiguous } XS \rangle$
shows $\langle \text{set_of_interval_list } XS = \text{set_of } (\text{interval_list_union } XS) \rangle$
 $\langle \text{proof} \rangle$

lemma width_eq_width_set:
fixes $X :: \langle 'a :: \{ \text{conditionally_complete_lattice, minus, preorder} \} \rangle \text{interval}$

shows $\langle \text{width } X = \text{width_set } (\text{set_of } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *width_zero_lower_upper*:
fixes $X :: \text{real interval}$
assumes $\langle \text{width } X = 0 \rangle$
shows $\langle \text{lower } X = \text{upper } X \rangle$
 $\langle \text{proof} \rangle$

lemma *width_zero_mk_interval*:
fixes $X :: \text{real interval}$
assumes $\langle \text{width } X = 0 \rangle$
shows $\langle \exists x. X = \text{mk_interval}(x,x) \rangle$
 $\langle \text{proof} \rangle$

lemma *width_zero_interval_of*:
fixes $X :: \text{real interval}$
assumes $\langle \text{width } X = 0 \rangle$
shows $\langle \exists x. X = \text{interval_of } x \rangle$
 $\langle \text{proof} \rangle$

lemma *width_zero_interval_extension*:
fixes $F :: \text{real interval} \Rightarrow \text{real interval}$
assumes $\langle F \text{ is_interval_extension_of } f \rangle$
and $\langle \text{width } X = 0 \rangle$
shows $\langle \text{width } (F X) = 0 \rangle$
 $\langle \text{proof} \rangle$

9.3 Lipschitz Interval Inclusive

If F is a natural interval extension of a real valued rational function with $F X$ defined for $X \subseteq Y$ where X and Y are intervals or n -dimensional interval vectors then F is Lipschitz in Y

lemma *interval_extension_bounded*:
fixes $F :: \text{real interval} \Rightarrow \text{real interval}$
assumes $\langle F \text{ is_interval_extension_of } f \rangle$
and $\langle (\text{width } (F X)) / (\text{width } X) \leq L \rangle$
shows $\text{width } (F X) \leq L * \text{width } X$
 $\langle \text{proof} \rangle$

lemma *lipschitz_on_implies_lipschitzl_on*:
fixes $F :: \text{real interval} \Rightarrow \text{real interval}$
assumes $\langle F \text{ is_interval_extension_of } f \rangle$
and $\langle C\text{-lipschitz_on } X f \rangle$
and $\langle \bigcup (\text{set_of } 'Y) \subseteq X \rangle$
and $\langle 0 \leq L \rangle$
and $\langle \forall y \in Y. (\text{width } (F y)) / (\text{width } y) \leq L \rangle$
shows $L\text{-lipschitzl_on } Y F$
 $\langle \text{proof} \rangle$

lemma *lipschitz_on_implies_lipschitzl_on2*:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $\langle S \neq [] \rangle$ **and** $\langle 0 \leq C \rangle$
and $\langle F \text{ is_interval_extension_of } f \rangle$

and $\langle 0 \leq L \rangle$
and $\langle \forall y \in (\text{set } S). (\text{width } (F y)) / (\text{width } y) \leq L \rangle$
and $\langle C\text{-lipschitz_on } (\text{set_of } (\text{interval_list_union } (S))) f \rangle$
shows $\langle L\text{-lipschitz_on } (\text{set } (S)) F \rangle$
 $\langle \text{proof} \rangle$

lemma *width_img_Max*:
assumes $\langle \text{finite } S \rangle$
shows $\langle \forall x \in S. \text{width } (F x) \leq \text{Max } (\text{width } 'F ' S) \rangle$
 $\langle \text{proof} \rangle$

lemma *width_Min*:
assumes $\langle \text{finite } S \rangle$
shows $\langle \forall x \in S. \text{Min } (\text{width } ' S) \leq \text{width } x \rangle$
 $\langle \text{proof} \rangle$

lemma *lipschitzl_on_le_interval*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inc_isotonic_}F: \langle \text{inclusion_isotonic } F \rangle$
and $\text{lipschitzl_}F: \langle C\text{-lipschitzl_on } \{X\} F \rangle$
and $\text{interval_inc}: \langle x \leq X \rangle$
shows $\langle \text{width } (F x) \leq C * \text{width } X \rangle$
 $\langle \text{proof} \rangle$

lemma *lipschitzl_on_le_lipschitzl_on*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inc_isotonic_}F: \langle \text{inclusion_isotonic } F \rangle$
and $\text{lipschitzl_}F: \langle C\text{-lipschitzl_on } \{X\} F \rangle$
and $\text{interval_inc}: \langle x \leq X \rangle$
and $\text{interval_ext}: \langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \exists L. L\text{-lipschitzl_on } \{x\} F \rangle$
 $\langle \text{proof} \rangle$

lemma *uniform_subdivision_le*:
fixes $X :: \langle \text{real interval} \rangle$
assumes $\langle N > 0 \rangle$
shows $\langle \forall x \in \text{set } (\text{uniform_subdivision } X N). x \leq X \rangle$
 $\langle \text{proof} \rangle$

lemma *lipschitzl_on_uniform_subdivision*:
fixes $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{inc_isotonic_}F: \langle \text{inclusion_isotonic } F \rangle$
and $\text{lipschitzl_}F: \langle C\text{-lipschitzl_on } (\{X\}) F \rangle$
and $\langle N > 0 \rangle$
shows $\langle \forall x \in (\text{set } (\text{uniform_subdivision } X N)). \text{width } (F x) \leq C * \text{width } X \rangle$
 $\langle \text{proof} \rangle$

lemma *division_leq_pos*:
fixes $x :: 'a :: \{\text{linordered_field}\}$
assumes $x > 0$ **and** $y > 0$ **and** $z > 0$ **and** $y \leq z$
shows $x / z \leq x / y$
 $\langle \text{proof} \rangle$

lemma *each_subdivision_width_order'*:

fixes $X :: \text{real interval}$
assumes $F \text{ is_interval_extension_of } f$
and $0 < N$
and $Xs \in \text{set } (\text{uniform_subdivision } X \ N)$
shows $\exists L. \text{width}(F \ Xs) \leq L * \text{width } (X) / \text{of_nat } N$
 $\langle \text{proof} \rangle$

lemma $\text{uniform_subdivision_min_nonzero}$:
assumes $\langle N > 0 \rangle$
and $\langle \text{width } X > 0 \rangle$
shows $\langle 0 < \text{Min } (\text{width } ' \text{set } (\text{uniform_subdivision } X \ N)) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{uniform_subdivision_width_zero_replicate_eq}$:
fixes $X :: \langle \text{real interval} \rangle$
assumes $\text{positive_}N: \langle 0 < N \rangle$
and $\text{zero_width_}X: \langle 0 = \text{width } X \rangle$
shows $\langle \text{replicate } N \ X = (\text{uniform_subdivision } X \ N) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{set_of_interval_list_zero_width}$:
fixes $X :: \langle \text{real interval} \rangle$
assumes $\text{positive_}N: \langle 0 < N \rangle$
and $\text{zero_width_}X: \langle 0 = \text{width } X \rangle$
shows $\langle \text{set_of_interval_list } (\text{uniform_subdivision } X \ N) = \{\text{lower } X .. \text{upper } X\} \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{width_zero_subdivision}$: $\text{width } X = (0 :: \text{real}) \implies N > 0 \implies \text{set } (\text{uniform_subdivision } X \ N) = \{X\}$
 $\langle \text{proof} \rangle$

lemma $\text{lipschitz_on_implies_lipschitzl_on_pre}$:
fixes $f :: \langle \text{real} \Rightarrow \text{real} \rangle$
and $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\langle \text{finite } S \rangle$
and $\langle 0 < \text{Min } (\text{width } ' S) \rangle$
shows $\langle \text{let } \text{max_}F_width = \text{Max } (\text{width } ' (F ' S));$
 $\text{min_}f_width = \text{Min } (\text{width } ' S)$
 $\text{in } \forall x \in S. \text{width } (F \ x) \leq (\text{max_}F_width / \text{min_}f_width) * \text{width } x \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{lipschitz_on_implies_lipschitzl_on}'$:
fixes $f :: \langle \text{real} \Rightarrow \text{real} \rangle$
and $F :: \langle \text{real interval} \Rightarrow \text{real interval} \rangle$
assumes $\text{non_empty}: \langle S \neq \{\} \rangle$
and $\text{finite}: \langle \text{finite } S \rangle$
and $\text{non_zero_width}: \langle 0 < \text{Min } (\text{width } ' S) \rangle$
and $\text{interval_ext_}F: \langle F \text{ is_interval_extension_of } f \rangle$
shows $\langle \exists L. L\text{-lipschitzl_on } S \ F \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{natural_extension_transfer_lipschitz}$:
assumes $\text{positive_}N: \langle 0 < N \rangle$
and $\text{inc_isontonic_}F: \langle \text{inclusion_isotonic } F \rangle$

and `interval_ext_F: <F is_natural_interval_extension_of f>`
and `lipschitz_f: <C—lipschitz_on (set_of X) f>`
shows `<C—lipschitzl_on (set (uniform_subdivision X N)) F>`
`<proof>`

lemma `lipschitz_on_division_lipschitz_on:`
assumes `lipschitz_f: C—lipschitz_on (set_of X) f`
and `non_empty: uniform_subdivision X N ≠ []`
and `subdivision: Xs ∈ set(uniform_subdivision (X::real interval) N)`
shows `∃ L . L—lipschitz_on (set_of Xs) f`
`<proof>`

lemma `lipschitz_on_lipschitz_on_subdivisions:`
assumes `lipschitz_f: C—lipschitz_on (set_of X) f`
and `non_empty: uniform_subdivision X N ≠ []`
and `non_zero: 0 < N`
shows `∃ L . ∀ Xs ∈ set(uniform_subdivision (X::real interval) N). L—lipschitz_on (set_of Xs) f`
`<proof>`

lemma `lipschitz_on_lipschitz_on_subdivisions_n:`
assumes `lipschitz_f: C—lipschitz_on (set_of X) f`
and `non_empty: uniform_subdivision X N ≠ []`
and `non_zero: 0 < N`
shows `∃ L . ∀ N > 0 . ∀ Xs ∈ set(uniform_subdivision (X::real interval) N). L—lipschitz_on (set_of Xs) f`
`<proof>`

lemma `lipschitzl_on_division_lipschitzl_on:`
assumes `lipschitz_f: C—lipschitzl_on (set(uniform_subdivision X N)) F`
and `non_empty: uniform_subdivision X N ≠ []`
and `subdivision: Xs ∈ set(uniform_subdivision (X::real interval) N)`
shows `∃ L . L—lipschitzl_on {Xs} F`
`<proof>`

lemma `lipschitzl_on_lipschitzl_on_subdivisions:`
fixes `X :: real interval`
assumes `lipschitz_f: C—lipschitzl_on (set(uniform_subdivision X N)) F`
and `non_zero: 0 < N`
shows `∃ L . ∀ Xs ∈ set(uniform_subdivision X N). L—lipschitzl_on {Xs} F`
`<proof>`

9.4 Lipschitz Convergence

lemma `isotonic_lipschitz_refinement':`
assumes `positive_N: <0 < N>`
and `inc_isotonic_F: <inclusion_isotonic F>`
and `interval_ext_F: <F is_interval_extension_of f>`
and `lipschitz_f: <C—lipschitz_on (set_of X) f>`
shows `<∃ L . width_set (set_of_interval_list (map F (uniform_subdivision X N))) — width_set (f' (set_of X)) ≤ 2 * (L * width X / real N)>`
`<proof>`

lemma `isotonic_lipschitz_refinementl:`

assumes *positive_N*: $\langle 0 < N \rangle$
and *inc_isotonic_F*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_ext_F*: $\langle F \text{ is_interval_extension_of } f \rangle$
and *lipschitz_f*: $\langle L\text{-lipschitz_on } (\text{set_of } X) f \rangle$
and *lipschitz_F*: $\langle C\text{-lipschitzl_on } (\text{set } (\text{uniform_subdivision } X N)) F \rangle$
shows $\langle \text{width_set } (\text{set_of_interval_list } (\text{map } F (\text{uniform_subdivision } X N))) - \text{width_set } (f' (\text{set_of } X)) \leq 2 * (C * \text{width } X / \text{real } N) \rangle$
<proof>

lemma *isotonic_lipschitz_refinement*:

assumes *positive_N*: $\langle 0 < N \rangle$
and *inc_isotonic_F*: $\langle \text{inclusion_isotonic } F \rangle$
and *interval_ext_F*: $\langle F \text{ is_interval_extension_of } f \rangle$
and *lipschitz_f*: $\langle L\text{-lipschitz_on } (\text{set_of } X) f \rangle$
and *lipschitz_F*: $\langle C\text{-lipschitzl_on } (\text{set } (\text{uniform_subdivision } X N)) F \rangle$
shows $\langle \text{excess_width_set } (\text{refinement_set } F N) f X \leq 2 * (C * \text{width } X / \text{real } N) \rangle$
<proof>

end

10 Interval Analysis (Interval_Analysis)

theory

Interval_Analysis

imports

Interval_Division_Real

Lipschitz_Subdivisions_Refnements

begin

This theory provides interval analysis over standard types such as real or integer. All operations work over (closed) intervals.

end

11 Extended Division on Intervals

(Extended_Interval_Division)

theory

Extended_Interval_Division

imports

Interval_Division_Non_Zero

begin

In this theory, we define an extended division operation on intervals. This definition is inspired by the definition given in [4], but we use an over-approximation for the case in which zero is an element of the divisor interval. By this, we avoid the need for multi-intervals.

instantiation *interval* :: (*{infinity, linordered_field, real_normed_algebra, linear_continuum_topology}*) *inverse*

begin

definition *inverse_interval* :: '*a interval* ⇒ '*a interval*

where *inverse_interval* *a* = (
 if ($\neg 0 \in_i a$) *then* *mk_interval* ($1 / (\text{upper } a), 1 / (\text{lower } a)$)
 else if $\text{lower } a = 0 \wedge 0 < \text{upper } a$ *then* *mk_interval* ($1 / \text{upper } a, \infty$)
 else if $\text{lower } a < 0 \wedge 0 < \text{upper } a$ *then* *mk_interval* ($-\infty, \infty$)
 else if $\text{lower } a < \text{upper } a \wedge \text{upper } a = 0$ *then* *mk_interval* ($-\infty, 1 / \text{lower } a$)
 else *undefined*
)

definition *divide_interval* :: '*a interval* ⇒ '*a interval* ⇒ '*a interval*

where *divide_interval* *a b* = *inverse* *b* * *a*

instance *<proof>*

end

interpretation *interval_division_inverse* *divide* *inverse*

<proof>

end

12 Extended Interval Analysis

(Extended_Interval_Analysis)

theory

Extended_Interval_Analysis

imports

Extended_Interval_Division

Lipschitz_Subdivisions_Refinements

begin

This theory provides extended interval analysis over the type extended reals. All operations work over (closed) intervals.

end

12.1 Overlapping Multi-Intervals (Multi_Interval_Overlapping)

theory

Multi_Interval_Overlapping

imports

Multi_Interval_Preliminaries

begin

12.1.1 Type Definition

```
typedef (overloaded) 'a minterval_ovl =
  {is::'a::{minus_mono} interval list. valid_mInterval_ovl is}
morphisms bounds_of_minterval_ovl mInterval_ovl
<proof>
```

```
setup_lifting type_definition_minterval_ovl
```

```
lift_definition mlower_ovl::('a::{minus_mono}) minterval_ovl  $\Rightarrow$  'a is <lower o hd> <proof>
```

```
lift_definition mupper_ovl::('a::{minus_mono}) minterval_ovl  $\Rightarrow$  'a is <upper o last> <proof>
```

```
lift_definition mlist_ovl::('a::{minus_mono}) minterval_ovl  $\Rightarrow$  'a interval list is <id> <proof>
```

12.1.2 Equality and Orderings

```
lemma minterval_ovl_eq_iff:  $a = b \iff$  mlist_ovl a = mlist_ovl b
<proof>
```

```
lemma ainterval_eqI: mlist_ovl a = mlist_ovl b  $\implies$  a = b
<proof>
```

```
lemma minterval_ovl_imp_upper_lower_eq :
  a = b  $\longrightarrow$  mlower_ovl a = mlower_ovl b  $\wedge$  mupper_ovl a = mupper_ovl b
<proof>
```

lemma `valid_mInterval_ovl_lower_le_upper`:
`valid_mInterval_ovl i \implies (lower \circ hd) i \leq (upper \circ last) i`
 `\langle proof \rangle`

lemma `mLower_non_ovl_le_mUpper_non_ovl[simp]`: `mLower_ovl i \leq mUpper_ovl i`
 `\langle proof \rangle`

lift_definition `set_of_ovl` :: `'a::minus_mono` `mInterval_ovl` \Rightarrow `'a set`
`is λ is. $\bigcup_{x \in \text{set } is}. \{\text{lower } x.. \text{upper } x\}$ \langle proof \rangle`

lemma `not_in_ovl_eq`:
 `\langle ($\neg e \in \text{set_of_ovl } xs$) = ($\forall x \in \text{set } (m\text{list_ovl } xs). \neg e \in \text{set_of } x$) \rangle`
 `\langle proof \rangle`

lemma `in_ovl_eq`:
 `\langle ($e \in \text{set_of_ovl } xs$) = ($\exists x \in \text{set } (m\text{list_ovl } xs). e \in \text{set_of } x$) \rangle`
 `\langle proof \rangle`

context notes `[[typedef_overloaded]] begin`

lift_definition `(code_dt) mInterval_ovl'` :: `'a::minus_mono` `interval list` \Rightarrow `'a mInterval_ovl option`
`is λ is. if valid_mInterval_ovl is then Some is else None`
 `\langle proof \rangle`

lemma `mInterval_ovl'_split`:
`P (mInterval_ovl' is) \longleftrightarrow`
`($\forall ivl. \text{valid_mInterval_ovl } is \longrightarrow m\text{list_ovl } ivl = is \longrightarrow P (\text{Some } ivl)$) \wedge ($\neg \text{valid_mInterval_ovl } is \longrightarrow P \text{None}$)`
 `\langle proof \rangle`

lemma `mInterval_ovl'_split_asm`:
`P (mInterval_ovl' is) \longleftrightarrow`
 `$\neg((\exists ivl. \text{valid_mInterval_ovl } is \wedge m\text{list_ovl } ivl = is \wedge \neg P (\text{Some } ivl)) \vee (\neg \text{valid_mInterval_ovl } is \wedge \neg P \text{None}))$`
 `\langle proof \rangle`

lemmas `mInterval_ovl'_splits = mInterval_ovl'_split mInterval_ovl'_split_asm`

lemma `mInterval'_eq_Some`: `mInterval_ovl' is = Some i \implies mlist_ovl i = is`
 `\langle proof \rangle`

end

lemma `set_of_ovl_non_zero_list_all`:
 `\langle 0 \notin set_of_ovl xs \implies $\forall x \in \text{set } (m\text{list_ovl } xs). \neg 0 \in_i x$ \rangle`
 `\langle proof \rangle`

instantiation `mInterval_ovl` :: `({minus_mono}) equal`
begin

definition `equal_class.equal` `a b \equiv (mList_ovl a = mList_ovl b)`

instance `\langle proof \rangle`
end

instantiation `mInterval_ovl` :: `({minus_mono}) ord` **begin**

definition *less_eq_minterval_ovl* :: 'a *minterval_ovl* \Rightarrow 'a *minterval_ovl* \Rightarrow bool
 where *less_eq_minterval_ovl* a b \longleftrightarrow *mlower_ovl* b \leq *mlower_ovl* a \wedge *mupper_ovl* a \leq *mupper_ovl* b

definition *less_minterval_ovl* :: 'a *minterval_ovl* \Rightarrow 'a *minterval_ovl* \Rightarrow bool
 where *less_minterval_ovl* x y = (x \leq y \wedge \neg y \leq x)

instance *<proof>*
end

instantiation *minterval_ovl* :: ({*minus_mono*,*lattice*}) *sup*
begin

lift_definition *sup_minterval_non_ovl* :: 'a *minterval_ovl* \Rightarrow 'a *minterval_ovl* \Rightarrow 'a *minterval_ovl*
is λ a b. [*Interval* (*inf* (*lower* (*hd* a)) (*lower* (*hd* b)), *sup* (*upper* (*last* a)) (*upper* (*last* b)))]
<proof>

instance
<proof>
end

instantiation *minterval_ovl* :: ({*lattice*,*minus_mono*}) *preorder*
begin
instance
<proof>
end

lift_definition *minterval_ovl_of* :: 'a::{*minus_mono*} \Rightarrow 'a *minterval_ovl* **is** λ x. [*Interval*(x, x)]
<proof>

lemma *mlower_ovl_minterval_ovl_of[simp]*: *mlower_ovl* (*minterval_ovl_of* a) = a
<proof>

lemma *mupper_ovl_minterval_ovl_of[simp]*: *mupper_ovl* (*minterval_ovl_of* a) = a
<proof>

definition *width_ovl* :: 'a::{*minus_mono*} *minterval_ovl* \Rightarrow 'a
 where *width_ovl* i = *mupper_ovl* i - *mlower_ovl* i

12.1.3 Zero and One

instantiation *minterval_ovl* :: ({*minus_mono*,*zero*}) *zero*
begin

lift_definition *zero_minterval_ovl*:: 'a *minterval_ovl* **is** *mk_mInterval_ovl* [*Interval* (o, o)]
<proof>

lemma *mlower_ovl_zero[simp]*: *mlower_ovl* o = o
<proof>

lemma *mupper_ovl_zero[simp]*: *mupper_ovl* o = o
<proof>

instance *<proof>*
end

instantiation *minterval_ovl* :: (*{minus_mono,one}*) *one*
begin

lift_definition *one_minterval_ovl*::'*a minterval_ovl is mk_mInterval_ovl [Interval (1, 1)]*
<proof>

lemma *mlower_ovl_one[simp]*: *mlower_ovl 1 = 1*
<proof>

lemma *mupper_ovl_one[simp]*: *mupper_ovl 1 = 1*
<proof>

instance *<proof>*
end

12.1.4 Addition

instantiation *minterval_ovl* :: (*{minus_mono,ordered_ab_semigroup_add,linordered_field}*) *plus*
begin

lift_definition *plus_minterval_ovl*::'*a minterval_ovl \Rightarrow 'a minterval_ovl \Rightarrow 'a minterval_ovl*
is $\lambda a b . mk_mInterval_ovl (iList_plus a b)$
<proof>

lemma *valid_mk_interval_iList_plus*:
assumes *valid_mInterval_ovl a* **and** *valid_mInterval_ovl b*
shows *valid_mInterval_ovl (mk_mInterval_ovl (iList_plus a b))*
<proof>

lift_definition *plus_minterval_non_ovl*::'*a minterval_ovl \Rightarrow 'a minterval_ovl \Rightarrow 'a minterval_ovl*
is $\lambda a b . mk_mInterval_ovl (iList_plus a b)$
<proof>

lemma *interval_plus_com*:
 *$\langle a + b = b + a \rangle$ for *a*::'*a*::*{minus_mono,ordered_ab_semigroup_add,linordered_field}* *minterval_ovl**
<proof>

instance *<proof>*
end

12.1.5 Unary Minus

lemma *a*: (*x*::'*a*::*ordered_ab_group_add interval*) $\neq y \implies -x \neq -y$
<proof>

lemma *b*: *distinct (is::'a::ordered_ab_group_add interval list) \implies distinct (map ($\lambda i . -i$) is)*
<proof>

instantiation *minterval_ovl* :: (*{minus_mono, ordered_ab_group_add}*) *uminus*
begin

lift_definition *uminus_minterval_ovl*::'*a minterval_ovl \Rightarrow 'a minterval_ovl*


```

    is  $\lambda$  is . mk_mInterval_ovl (rev (map ( $\lambda$  i.  $-i$ ) is))
  <proof>
instance <proof>
end

```

12.1.6 Subtraction

```

instantiation minterval_ovl :: ({minus_mono, linordered_field, ordered_ab_group_add}) minus
begin

```

```

definition minus_minterval_ovl :: 'a minterval_ovl  $\Rightarrow$  'a minterval_ovl  $\Rightarrow$  'a minterval_ovl
  where minus_minterval_ovl a b = a + - b

```

```

instance <proof>
end

```

12.1.7 Multiplication

```

instantiation minterval_ovl :: ({minus_mono, linordered_semiring}) times
begin

```

```

lift_definition times_minterval_ovl :: 'a minterval_ovl  $\Rightarrow$  'a minterval_ovl  $\Rightarrow$  'a minterval_ovl
  is  $\lambda$  a b . mk_mInterval_ovl (iList_times a b)
  <proof>

```

```

instance <proof>
end

```

12.1.8 Multiplicative Inverse and Division

```

locale minterval_ovl_division = inverse +
  constrains inverse :: <'a:: {linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}
  minterval_ovl  $\Rightarrow$  'a minterval_ovl
  and divide :: <'a:: {linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}
  minterval_ovl  $\Rightarrow$  'a minterval_ovl  $\Rightarrow$  'a minterval_ovl
  assumes inverse_left:  $\langle \neg 0 \in \text{set\_of\_ovl } x \implies 1 \leq (\text{inverse } x) * x \rangle$ 
  and divide:  $\langle \neg 0 \in \text{set\_of\_ovl } y \implies x \leq (\text{divide } x y) * y \rangle$ 
begin
end

```

```

locale minterval_ovl_division_inverse = inverse +
  constrains inverse :: <'a:: {linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}
  minterval_ovl  $\Rightarrow$  'a minterval_ovl
  and divide :: <'a:: {linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}
  minterval_ovl  $\Rightarrow$  'a minterval_ovl  $\Rightarrow$  'a minterval_ovl
  assumes inverse_non_zero_def:  $\langle \neg 0 \in \text{set\_of\_ovl } x \implies (\text{inverse } x)
    = \text{mInterval\_ovl } (\text{mk\_mInterval\_ovl}(\text{un\_op\_interval\_list } (\lambda$  i.  $\text{mk\_interval } (1 / (\text{upper } i), 1 / (\text{lower } i))))
    (\text{mList\_ovl } x)) \rangle$ 
  and divide_non_zero_def:  $\langle \neg 0 \in \text{set\_of\_ovl } y \implies (\text{divide } x y) = \text{inverse } y * x \rangle$ 
begin
end

```

12.1.9 Membership

abbreviation (in *preorder*) *in_minterval_ovl* (($_ / \in_{no}$) [51, 51] 50)
where *in_minterval_ovl* $x X \equiv x \in \text{set_of_ovl } X$

lemma *in_minterval_ovl_to_minterval_ovl*[*intro!*]: $a \in_{no} \text{minterval_ovl_of } a$
<*proof*>

instance *minterval_ovl* :: ($\{one, preorder, minus_mono, linordered_semiring\}$) *power*
<*proof*>

lemma *set_of_one_ovl*[*simp*]: $\text{set_of_ovl } (1::'a::\{one, order, minus_mono\} \text{minterval_ovl}) = \{1\}$
<*proof*>

lifting_update *minterval_ovl.lifting*

lifting_forget *minterval_ovl.lifting*

end

12.2 Non-Overlapping Multi-Intervals (\exists Multi_Interval_Non_Overlapping)

theory

Multi_Interval_Non_Overlapping

imports

Multi_Interval_Preliminaries

begin

12.2.1 Type Definition

typedef (**overloaded**) $'a \text{minterval_non_ovl} =$
 $\{is::'a::\{minus_mono\} \text{interval list. valid_mInterval_non_ovl } is\}$
morphisms *bounds_of_minterval_non_ovl* *mInterval_non_ovl*
<*proof*>

setup_lifting *type_definition_minterval_non_ovl*

lift_definition *mlower_non_ovl*::($'a::\{minus_mono\}$) *minterval_non_ovl* $\Rightarrow 'a \text{ is } \langle \text{lower o hd} \rangle$ <*proof*>

lift_definition *mupper_non_ovl*::($'a::\{minus_mono\}$) *minterval_non_ovl* $\Rightarrow 'a \text{ is } \langle \text{upper o last} \rangle$ <*proof*>

lift_definition *mlist_non_ovl*::($'a::\{minus_mono\}$) *minterval_non_ovl* $\Rightarrow 'a \text{ interval list is } \langle \text{id} \rangle$ <*proof*>

12.2.2 Equality and Orderings

lemma *minterval_non_ovl_eq_iff*: $a = b \iff \text{mlist_non_ovl } a = \text{mlist_non_ovl } b$
<*proof*>

lemma *ainterval_eq*: $\text{mlist_non_ovl } a = \text{mlist_non_ovl } b \implies a = b$
<*proof*>

lemma *minterval_non_ovl_imp_upper_lower_eq* :
 $a = b \implies \text{mlower_non_ovl } a = \text{mlower_non_ovl } b \wedge \text{mupper_non_ovl } a = \text{mupper_non_ovl } b$
<*proof*>

lemma `mlower_non_ovl_le_mupper_non_ovl[simp]`: $mlower_non_ovl\ i \leq mupper_non_ovl\ i$
<proof>

lift_definition `set_of_non_ovl` :: $'a::\{minus_mono\}$ `mininterval_non_ovl` \Rightarrow $'a$ set
is λ is. $\bigcup_{x \in set\ is} \{lower\ x..upper\ x\}$ *<proof>*

lemma `set_non_ovl_of_subset`: $set_of_non_ovl\ (x::'a::minus_mono\ mininterval_non_ovl) \subseteq \{mlower_non_ovl\ x .. mupper_non_ovl\ x\}$
<proof>

lemma `not_in_non_ovl_eq`:
 $\langle \neg e \in set_of_non_ovl\ xs \rangle = (\forall x \in set\ (mlist_non_ovl\ xs). \neg e \in set_of\ x)$
<proof>

lemma `in_non_ovl_eq`:
 $\langle e \in set_of_non_ovl\ xs \rangle = (\exists x \in set\ (mlist_non_ovl\ xs). e \in set_of\ x)$
<proof>

lemma `set_of_non_ovl_non_zero_list_all`:
 $\langle 0 \notin set_of_non_ovl\ xs \implies \forall x \in set\ (mlist_non_ovl\ xs). \neg 0 \in_i\ x \rangle$
<proof>

context notes `[[typedef_overloaded]] begin`

lift_definition `(code_dt) mInterval_non_ovl'` :: $'a::minus_mono$ `interval list` \Rightarrow $'a$ `mininterval_non_ovl option`
is λ is. if valid_mInterval_non_ovl is then Some is else None
<proof>

lemma `mInterval_non_ovl'_split`:
 $P\ (mInterval_non_ovl'\ is) \longleftrightarrow$
 $(\forall ivl. valid_mInterval_non_ovl\ is \longrightarrow mlist_non_ovl\ ivl = is \longrightarrow P\ (Some\ ivl)) \wedge (\neg valid_mInterval_non_ovl\ is \longrightarrow P\ None)$
<proof>

lemma `mInterval_non_ovl'_split_asm`:
 $P\ (mInterval_non_ovl'\ is) \longleftrightarrow$
 $\neg((\exists ivl. valid_mInterval_non_ovl\ is \wedge mlist_non_ovl\ ivl = is \wedge \neg P\ (Some\ ivl)) \vee (\neg valid_mInterval_non_ovl\ is \wedge \neg P\ None))$
<proof>

lemmas `mInterval_non_ovl'_splits = mInterval_non_ovl'_split mInterval_non_ovl'_split_asm`

lemma `mInterval'_eq_Some`: $mInterval_non_ovl'\ is = Some\ i \implies mlist_non_ovl\ i = is$
<proof>

end

instantiation `mininterval_non_ovl` :: $(\{minus_mono\})$ equal
begin

definition `equal_class.equal` $a\ b \equiv (mlist_non_ovl\ a = mlist_non_ovl\ b)$

instance $\langle proof \rangle$
end

instantiation $minterval_non_ovl :: (\{minus_mono\}) \text{ ord } \text{begin}$

definition $less_eq_minterval_non_ovl :: 'a \text{ minterval_non_ovl} \Rightarrow 'a \text{ minterval_non_ovl} \Rightarrow \text{bool}$
where $less_eq_minterval_non_ovl \ a \ b \iff mlower_non_ovl \ b \leq mlower_non_ovl \ a \wedge mupper_non_ovl \ a \leq mupper_non_ovl \ b$

definition $less_minterval_non_ovl :: 'a \text{ minterval_non_ovl} \Rightarrow 'a \text{ minterval_non_ovl} \Rightarrow \text{bool}$
where $less_minterval_non_ovl \ x \ y = (x \leq y \wedge \neg y \leq x)$

instance $\langle proof \rangle$
end

instantiation $minterval_non_ovl :: (\{minus_mono, lattice\}) \text{ sup } \text{begin}$

lift_definition $sup_minterval_non_ovl :: 'a \text{ minterval_non_ovl} \Rightarrow 'a \text{ minterval_non_ovl} \Rightarrow 'a \text{ minterval_non_ovl}$
is $\lambda \ a \ b. [Interval \ (inf \ (lower \ (hd \ a)) \ (lower \ (hd \ b))), \ sup \ (upper \ (last \ a)) \ (upper \ (last \ b))]$
 $\langle proof \rangle$

lemma $mlower_non_ovl_sup[simp]: mlower_non_ovl \ (sup \ A \ B) = inf \ (mlower_non_ovl \ A) \ (mlower_non_ovl \ B)$
 $\langle proof \rangle$

lemma $mupper_non_ovl_sup[simp]: mupper_non_ovl \ (sup \ A \ B) = sup \ (mupper_non_ovl \ A) \ (mupper_non_ovl \ B)$
 $\langle proof \rangle$

instance
 $\langle proof \rangle$
end

instantiation $minterval_non_ovl :: (\{lattice, minus_mono\}) \text{ preorder}$

begin
instance
 $\langle proof \rangle$
end

lemma $set_of_minterval_non_ovl_union: set_of_non_ovl \ A \cup set_of_non_ovl \ B \subseteq set_of_non_ovl \ (sup \ A \ B)$
for $A :: 'a :: \{lattice, minus_mono\} \text{ minterval_non_ovl}$
 $\langle proof \rangle$

lemma $minterval_non_ovl_union_commute: sup \ A \ B = sup \ B \ A$ **for** $A :: 'a :: \{minus_mono, lattice\} \text{ minterval_non_ovl}$
 $\langle proof \rangle$

lemma $minterval_non_ovl_union_mono1: set_of_non_ovl \ a \subseteq set_of_non_ovl \ (sup \ a \ A)$
for $A :: 'a :: \{minus_mono, lattice\} \text{ minterval_non_ovl}$
 $\langle proof \rangle$

lemma $minterval_non_ovl_union_mono2: set_of_non_ovl \ A \subseteq set_of_non_ovl \ (sup \ a \ A)$ **for** $A :: 'a :: \{lattice, minus_mono\} \text{ minterval_non_ovl}$
 $\langle proof \rangle$

lift_definition $minterval_non_ovl_of :: 'a :: \{minus_mono\} \Rightarrow 'a \text{ minterval_non_ovl } \text{is } \lambda x. [Interval(x, x)]$

<proof>

lemma *mlower_non_ovl_minterval_non_ovl_of*[simp]: $mlower_non_ovl (minterval_non_ovl_of\ a) = a$
<proof>

lemma *mupper_non_ovl_minterval_non_ovl_of*[simp]: $mupper_non_ovl (minterval_non_ovl_of\ a) = a$
<proof>

definition *width_non_ovl* :: 'a::{minus_mono} *minterval_non_ovl* \Rightarrow 'a
where $width_non_ovl\ i = mupper_non_ovl\ i - mlower_non_ovl\ i$

12.2.3 Zero and One

instantiation *minterval_non_ovl* :: ({minus_mono,zero}) zero
begin

lift_definition *zero_minterval_non_ovl*::'a *minterval_non_ovl* is *mk_mInterval_non_ovl* [Interval (0, 0)]
<proof>

lemma *mlower_non_ovl_zero*[simp]: $mlower_non_ovl\ 0 = 0$
<proof>

lemma *mupper_non_ovl_zero*[simp]: $mupper_non_ovl\ 0 = 0$
<proof>

instance *<proof>*
end

instantiation *minterval_non_ovl* :: ({minus_mono,one}) one
begin

lift_definition *one_minterval_non_ovl*::'a *minterval_non_ovl* is *mk_mInterval_non_ovl* [Interval (1, 1)]
<proof>

lemma *mlower_non_ovl_one*[simp]: $mlower_non_ovl\ 1 = 1$
<proof>

lemma *mupper_non_ovl_one*[simp]: $mupper_non_ovl\ 1 = 1$
<proof>

instance *<proof>*
end

12.2.4 Addition

instantiation *minterval_non_ovl* :: ({minus_mono,ordered_ab_semigroup_add,linordered_field}) plus
begin

lemma *valid_mk_interval_iList_plus*:
assumes *valid_mInterval_non_ovl a* and *valid_mInterval_non_ovl b*
shows *valid_mInterval_non_ovl (mk_mInterval_non_ovl (iList_plus a b))*
<proof>

lift_definition *plus_minterval_non_ovl*::'a *minterval_non_ovl* \Rightarrow 'a *minterval_non_ovl* \Rightarrow 'a *minterval_non_ovl*

```
is λ a b . mk_mInterval_non_ovl (iList_plus a b)
⟨proof⟩
```

lemma *interval_plus_com*:

```
⟨a + b = b + a⟩ for a::'a::{minus_mono,ordered_ab_semigroup_add,linordered_field} minterval_non_ovl
⟨proof⟩
```

instance ⟨proof⟩

end

12.2.5 Unary Minus

lemma *a*: (x::'a::ordered_ab_group_add interval) ≠ y ⇒ -x ≠ -y
⟨proof⟩

lemma *b*: distinct (is::'a::ordered_ab_group_add interval list) ⇒ distinct (map (λ i. -i) is)
⟨proof⟩

instantiation *minterval_non_ovl* :: ({minus_mono, ordered_ab_group_add}) uminus
begin

lift_definition *uminus_minterval_non_ovl*::'a minterval_non_ovl ⇒ 'a minterval_non_ovl
is λ is . mk_mInterval_non_ovl (rev (map (λ i. -i) is))
⟨proof⟩

instance ⟨proof⟩

end

12.2.6 Subtraction

instantiation *minterval_non_ovl* :: ({minus_mono, linordered_field, ordered_ab_group_add}) minus
begin

definition *minus_minterval_non_ovl*::'a minterval_non_ovl ⇒ 'a minterval_non_ovl ⇒ 'a minterval_non_ovl
where *minus_minterval_non_ovl* a b = a + - b

instance ⟨proof⟩

end

12.2.7 Multiplication

instantiation *minterval_non_ovl* :: ({minus_mono, linordered_field}) times
begin

lift_definition *times_minterval_non_ovl*::'a minterval_non_ovl ⇒ 'a minterval_non_ovl ⇒ 'a minterval_non_ovl
is λ a b . mk_mInterval_non_ovl (iList_times a b)
⟨proof⟩

instance ⟨proof⟩

end

12.2.8 Multiplicative Inverse and Division

```
locale minterval_non_ovl_division = inverse +  
  constrains inverse :: <'a::{\linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}  
minterval_non_ovl ⇒ 'a minterval_non_ovl  
    and divide :: <'a::{\linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}  
minterval_non_ovl ⇒ 'a minterval_non_ovl ⇒ 'a minterval_non_ovl  
    assumes inverse_left: <¬ 0 ∈ set_of_non_ovl x ⇒ 1 ≤ (inverse x) * x>  
    and divide: <¬ 0 ∈ set_of_non_ovl y ⇒ x ≤ (divide x y) * y>  
begin  
end
```

```
locale minterval_non_ovl_division_inverse = inverse +  
  constrains inverse :: <'a::{\linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}  
minterval_non_ovl ⇒ 'a minterval_non_ovl  
    and divide :: <'a::{\linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology}  
minterval_non_ovl ⇒ 'a minterval_non_ovl ⇒ 'a minterval_non_ovl  
    assumes inverse_non_zero_def: <¬ 0 ∈ set_of_non_ovl x ⇒ (inverse x)  
      = mInterval_non_ovl (mk_mInterval_non_ovl(un_op_interval_list (λ i. mk_interval (1 / (upper i), 1 /  
(lower i))) (mList_non_ovl x)))>  
    and divide_non_zero_def: <¬ 0 ∈ set_of_non_ovl y ⇒ (divide x y) = inverse y * x>  
begin  
end
```

12.2.9 Membership

```
abbreviation (in preorder) in_minterval_non_ovl ((_/ ∈n.o.) [51, 51] 50)  
where in_minterval_non_ovl x X ≡ x ∈ set_of_non_ovl X
```

```
lemma in_minterval_non_ovl_to_minterval_non_ovl[intro!]: a ∈n.o. minterval_non_ovl of a  
  <proof>
```

```
instance minterval_non_ovl :: ({one, preorder, minus_mono, linordered_semiring}) power  
  <proof>
```

```
lemma set_of_one_non_ovl[simp]: set_of_non_ovl (1::'a::{\one, minus_mono, order} minterval_non_ovl) = {1}  
  <proof>
```

```
lifting_update minterval_non_ovl.lifting  
lifting_forget minterval_non_ovl.lifting
```

end

12.3 Adjacent Multi-Intervals (Multi_Interval_Adjacent)

```
theory  
  Multi_Interval_Adjacent  
imports  
  Multi_Interval_Preliminaries  
begin
```

12.3.1 A Type For Non Overlapping Multi Intervals

typedef (overloaded) 'a minterval_adj =
{is::'a::{minus_mono,linorder} interval list. valid_mInterval_adj is}
morphisms bounds_of_minterval_adj mInterval_adj
<proof>

setup_lifting type_definition_minterval_adj

lift_definition mlower_adj::('a::{minus_mono}) minterval_adj \Rightarrow 'a is <lower o hd> <proof>

lift_definition mupper_adj::('a::{minus_mono}) minterval_adj \Rightarrow 'a is <upper o last> <proof>

lift_definition mlist_adj::('a::{minus_mono}) minterval_adj \Rightarrow 'a interval list is <id> <proof>

12.3.2 Equality and Orderings

lemma minterval_adj_eq_iff: $a = b \iff mlist_adj\ a = mlist_adj\ b$
<proof>

lemma ainterval_eq1: $mlist_adj\ a = mlist_adj\ b \implies a = b$
<proof>

lemma minterval_adj_imp_upper_lower_eq :
 $a = b \implies mlower_adj\ a = mlower_adj\ b \wedge mupper_adj\ a = mupper_adj\ b$
<proof>

lemma mlower_adj_le_mupper_adj[simp]: $mlower_adj\ i \leq mupper_adj\ i$
<proof>

lift_definition set_of_adj :: 'a::{minus_mono} minterval_adj \Rightarrow 'a set
is λ is. $\bigcup_{x \in \text{set is.}} \{lower\ x..upper\ x\}$ <proof>

lemma set_adj_of_subset: $\text{set_of_adj}\ (x::'a::\text{minus_mono}\ \text{minterval_adj}) \subseteq \{mlower_adj\ x .. mupper_adj\ x\}$
<proof>

lemma not_in_adj_eq:
 $\langle \neg e \in \text{set_of_adj}\ xs \rangle = (\forall x \in \text{set}\ (mlist_adj\ xs). \neg e \in \text{set_of}\ x)$
<proof>

lemma in_adj_eq:
 $\langle e \in \text{set_of_adj}\ xs \rangle = (\exists x \in \text{set}\ (mlist_adj\ xs). e \in \text{set_of}\ x)$
<proof>

lemma set_of_adj_non_zero_list_all:
 $\langle 0 \notin \text{set_of_adj}\ xs \implies \forall x \in \text{set}\ (mlist_adj\ xs). \neg 0 \in_i\ x \rangle$
<proof>

context notes [[typedef_overloaded]] **begin**

lift_definition(code_dt) mInterval_adj'::'a::minus_mono interval list \Rightarrow 'a minterval_adj option
is λ is. if valid_mInterval_adj is then Some is else None
<proof>

lemma *mInterval_adj'_split*:
 $P (mInterval_adj' is) \longleftrightarrow$
 $(\forall ivl. valid_mInterval_adj is \longrightarrow mlist_adj ivl = is \longrightarrow P (Some ivl)) \wedge (\neg valid_mInterval_adj is \longrightarrow P None)$
 $\langle proof \rangle$

lemma *mInterval_adj'_split_asm*:
 $P (mInterval_adj' is) \longleftrightarrow$
 $\neg((\exists ivl. valid_mInterval_adj is \wedge mlist_adj ivl = is \wedge \neg P (Some ivl)) \vee (\neg valid_mInterval_adj is \wedge \neg P None))$
 $\langle proof \rangle$

lemmas *mInterval_adj'_splits = mInterval_adj'_split mInterval_adj'_split_asm*

lemma *mInterval'_eq_Some*: $mInterval_adj' is = Some i \implies mlist_adj i = is$
 $\langle proof \rangle$

end

instantiation *minterval_adj* :: ($\{minus_mono\}$) *equal*
begin

definition *equal_class.equal* $a b \equiv (mlist_adj a = mlist_adj b)$

instance $\langle proof \rangle$
end

instantiation *minterval_adj* :: ($\{minus_mono\}$) *ord* **begin**

definition *less_eq_minterval_adj* :: $'a \text{ minterval_adj} \Rightarrow 'a \text{ minterval_adj} \Rightarrow \text{bool}$
where $less_eq_minterval_adj a b \longleftrightarrow mlower_adj b \leq mlower_adj a \wedge mupper_adj a \leq mupper_adj b$

definition *less_minterval_adj* :: $'a \text{ minterval_adj} \Rightarrow 'a \text{ minterval_adj} \Rightarrow \text{bool}$
where $less_minterval_adj x y = (x \leq y \wedge \neg y \leq x)$

instance $\langle proof \rangle$
end

instantiation *minterval_adj* :: ($\{minus_mono, lattice\}$) *sup*
begin

lift_definition *sup_minterval_adj* :: $'a \text{ minterval_adj} \Rightarrow 'a \text{ minterval_adj} \Rightarrow 'a \text{ minterval_adj}$
is $\lambda a b. [Interval (inf (lower (hd a)) (lower (hd b))), sup (upper (last a)) (upper (last b))]$
 $\langle proof \rangle$

lemma *mlower_adj_sup[simp]*: $mlower_adj (sup A B) = inf (mlower_adj A) (mlower_adj B)$
 $\langle proof \rangle$

lemma *mupper_adj_sup[simp]*: $mupper_adj (sup A B) = sup (mupper_adj A) (mupper_adj B)$
 $\langle proof \rangle$

instance
 $\langle proof \rangle$
end

instantiation *minterval_adj* :: ($\{lattice, minus_mono\}$) *preorder*
begin

instance

<proof>

end

lemma *set_of_minterval_adj_union*: $\text{set_of_adj } A \cup \text{set_of_adj } B \subseteq \text{set_of_adj } (\text{sup } A \ B)$

for $A :: 'a :: \{\text{lattice, minus_mono}\}$ *mininterval_adj*

<proof>

lemma *mininterval_adj_union_commute*: $\text{sup } A \ B = \text{sup } B \ A$ **for** $A :: 'a :: \{\text{minus_mono, lattice}\}$ *mininterval_adj*

<proof>

lemma *mininterval_adj_union_mono1*: $\text{set_of_adj } a \subseteq \text{set_of_adj } (\text{sup } a \ A)$

for $A :: 'a :: \{\text{minus_mono, lattice}\}$ *mininterval_adj*

<proof>

lemma *mininterval_adj_union_mono2*: $\text{set_of_adj } A \subseteq \text{set_of_adj } (\text{sup } a \ A)$ **for** $A :: 'a :: \{\text{lattice, minus_mono}\}$ *mininterval_adj*

<proof>

lift_definition *mininterval_adj_of* :: $'a :: \{\text{minus_mono}\} \Rightarrow 'a \ \text{mininterval_adj}$ **is** $\lambda x. [\text{Interval}(x, x)]$

<proof>

lemma *mlower_adj_mininterval_adj_of[simp]*: $\text{mlower_adj } (\text{mininterval_adj_of } a) = a$

<proof>

lemma *mupper_adj_mininterval_adj_of[simp]*: $\text{mupper_adj } (\text{mininterval_adj_of } a) = a$

<proof>

definition *width_adj* :: $'a :: \{\text{minus_mono}\}$ *mininterval_adj* $\Rightarrow 'a$

where $\text{width_adj } i = \text{mupper_adj } i - \text{mlower_adj } i$

12.3.3 Zero and One

instantiation *mininterval_adj* :: $(\{\text{minus_mono, zero}\})$ *zero*

begin

lift_definition *zero_mininterval_adj*:: $'a \ \text{mininterval_adj}$ **is** $\text{mk_mInterval_adj } [\text{Interval } (0, 0)]$

<proof>

lemma *mlower_adj_zero[simp]*: $\text{mlower_adj } 0 = 0$

<proof>

lemma *mupper_adj_zero[simp]*: $\text{mupper_adj } 0 = 0$

<proof>

instance *<proof>*

end

instantiation *mininterval_adj* :: $(\{\text{minus_mono, one}\})$ *one*

begin

lift_definition *one_mininterval_adj*:: $'a \ \text{mininterval_adj}$ **is** $\text{mk_mInterval_adj } [\text{Interval } (1, 1)]$

<proof>

lemma *mlower_adj_one[simp]*: $\text{mlower_adj } 1 = 1$

<proof>

lemma *mupper_adj_one*[simp]: *mupper_adj* 1 = 1

<proof>

instance *<proof>*

end

12.3.4 Addition

instantiation *minterval_adj* :: (*{minus_mono, ordered_ab_semigroup_add, linordered_field}*) *plus*
begin

lift_definition *plus_minterval_adj*::*'a minterval_adj* \Rightarrow *'a minterval_adj* \Rightarrow *'a minterval_adj*

is $\lambda a b . mk_mInterval_adj (iList_plus a b)$

<proof>

instance *<proof>*

lemma *interval_plus_com*:

$\langle a + b = b + a \rangle$ **for** *a*::*'a minterval_adj*

<proof>

end

12.3.5 Unary Minus

lemma *a*: (*x*::*'a::ordered_ab_group_add interval*) $\neq y \Rightarrow -x \neq -y$

<proof>

lemma *b*: *distinct (is::'a::ordered_ab_group_add interval list)* \Rightarrow *distinct (map ($\lambda i . -i$) is)*

<proof>

instantiation *minterval_adj* :: (*{minus_mono, ordered_ab_group_add}*) *uminus*

begin

lift_definition *uminus_minterval_non_ovl*::*'a minterval_adj* \Rightarrow *'a minterval_adj*

is $\lambda is . mk_mInterval_non_ovl (rev (map ($\lambda i . -i$) is))$

<proof>

instance *<proof>*

end

12.3.6 Subtraction

instantiation *minterval_adj* :: (*{minus_mono, linordered_field, ordered_ab_group_add}*) *minus*
begin

definition *minus_minterval_non_ovl*::*'a minterval_adj* \Rightarrow *'a minterval_adj* \Rightarrow *'a minterval_adj*

where *minus_minterval_non_ovl a b = a + - b*

instance *<proof>*

end

12.3.7 Multiplication

instantiation *minterval_adj* :: (*{minus_mono, linordered_field}*) *times*
begin

lift_definition *times_minterval_non_ovl*:: 'a *minterval_adj* \Rightarrow 'a *minterval_adj* \Rightarrow 'a *minterval_adj*
is $\lambda a b . mk_minterval_non_ovl (iList_times a b)$
<proof>

instance *<proof>*
end

12.3.8 Multiplicative Inverse and Division

locale *minterval_adj_division* = *inverse* +
constrains *inverse* :: 'a::(*linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology*)
minterval_adj \Rightarrow 'a *minterval_adj*
and *divide* :: 'a::(*linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology*)
minterval_adj \Rightarrow 'a *minterval_adj*
assumes *inverse_left*: $\neg 0 \in set_of_adj\ x \implies 1 \leq (inverse\ x) * x$
and *divide*: $\neg 0 \in set_of_adj\ y \implies x \leq (divide\ x\ y) * y$
begin
end

locale *minterval_adj_division_inverse* = *inverse* +
constrains *inverse* :: 'a::(*linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology*)
minterval_adj \Rightarrow 'a *minterval_adj*
and *divide* :: 'a::(*linordered_field, zero, minus, minus_mono, real_normed_algebra, linear_continuum_topology*)
minterval_adj \Rightarrow 'a *minterval_adj* \Rightarrow 'a *minterval_adj*
assumes *inverse_non_zero_def*: $\neg 0 \in set_of_adj\ x \implies (inverse\ x)$
 $= mInterval_adj (mk_minterval_adj (un_op_interval_list (\lambda i . mk_interval (1 / (upper\ i), 1 / (lower\ i))))$
 $(mList_adj\ x))$
and *divide_non_zero_def*: $\neg 0 \in set_of_adj\ y \implies (divide\ x\ y) = inverse\ y * x$
begin
end

12.3.9 Membership

abbreviation (**in** *preorder*) *in_minterval_adj* ((*_* / \in_{adj} *_*) [51, 51] 50)
where *in_minterval_adj* *x X* $\equiv x \in set_of_adj\ X$

lemma *in_minterval_adj_to_minterval_adj[intro!]*: $a \in_{adj}\ minterval_adj_of\ a$
<proof>

instance *minterval_adj* :: (*{one, preorder, minus_mono, linordered_semiring}*) *power*
<proof>

lemma *set_of_one_adj[simp]*: $set_of_adj (1::'a::\{one, minus_mono, order\}\ minterval_adj) = \{1\}$
<proof>

lifting_update *minterval_adj.lifting*
lifting_forget *minterval_adj.lifting*

end

12.4 Bringing Everything Together (📄 Multi_Interval)

theory

Multi_Interval

imports

Multi_Interval_Overlapping

Multi_Interval_Non_Overlapping

Multi_Interval_Adjacent

Lipschitz_Subdivisions_Refinements

begin

For convince, we provide a theory that provides all three variants of multi-intervals. Note that the order in which these theories are imported is important: importing the theory `Multi_Interval_Adjacent` last ensures that it provides the default definitions and lemmas.

end

13 Extended Division on Multi-Intervals

(Extended_Multi_Interval_Division_Core)

theory

Extended_Multi_Interval_Division_Core

imports

Interval_Division_Non_Zero

Multi_Interval

begin

13.1 Division over List of Intervals

In this theory, we define an extended division operation on intervals. This is a formalization of the interval division given in [4].

definition *inverse_interval* :: ('a::{'linorder, minus_mono, zero, one, inverse, infinity, uminus'}) interval \Rightarrow ('a interval) list
where *inverse_interval* a = (

if $(\neg o \in_i a)$ then [mk_interval (1 / (upper a), 1 / (lower a))]
 else if lower a = o \wedge o < upper a then [mk_interval (1 / upper a, ∞)]
 else if lower a < o \wedge o < upper a then [mk_interval ($-\infty$, 1 / lower a), mk_interval (1 / upper a, ∞)]
 else if lower a < upper a \wedge upper a = o then [mk_interval ($-\infty$, 1 / lower a)]
 else undefined

)

definition \langle minverse = concat o (map inverse_interval) \rangle

13.2 Multi-Interval Division

end

13.2.1 Overlapping Multi-Intervals (Extended_Multi_Interval_Division_Overlapping)

theory

Extended_Multi_Interval_Division_Overlapping

imports

Extended_Multi_Interval_Division_Core

begin

definition \langle mininterval_ovl_inverse x = mInterval_ovl (mk_mInterval_ovl(minverse (mList_ovl x))) \rangle

definition \langle mininterval_ovl_divide x y = (mininterval_ovl_inverse y) * x \rangle

lemma set_of_ovl_non_zero_map_inverse:

assumes $\langle o \notin \text{set_of_ovl } xs \rangle$

shows $\langle \text{concat (map inverse_interval (mList_ovl } xs)) = \text{map } (\lambda i. \text{mk_interval (1 / upper } i, 1 / \text{lower } i)) \text{ (mList_ovl } xs) \rangle$

\langle proof \rangle

interpretation mininterval_ovl_division_inverse mininterval_ovl_divide mininterval_ovl_inverse

\langle proof \rangle

end

13.2.2 Non Overlapping Multi-Intervals (Extended_Multi_Interval_Division_Non_Overlapping)

theory

Extended_Multi_Interval_Division_Non_Overlapping

imports

Extended_Multi_Interval_Division_Core

begin

definition $\langle \text{minterval_non_ovl_inverse } x = \text{mInterval_non_ovl } (\text{mk_mInterval_non_ovl } (\text{minverse } (\text{mlist_non_ovl } x))) \rangle$

definition $\langle \text{minterval_non_ovl_divide } x y = (\text{minterval_non_ovl_inverse } y) * x \rangle$

lemma *set_of_non_ovl_non_zero_map_inverse:*

assumes $\langle 0 \notin \text{set_of_non_ovl } xs \rangle$

shows $\langle \text{concat } (\text{map } \text{inverse_interval } (\text{mlist_non_ovl } xs)) = \text{map } (\lambda i. \text{mk_interval } (1 / \text{upper } i, 1 / \text{lower } i)) (\text{mlist_non_ovl } xs) \rangle$

<proof>

interpretation *minterval_non_ovl_division_inverse minterval_non_ovl_divide minterval_non_ovl_inverse*

<proof>

end

13.2.3 Adjacent Multi-Intervals (Extended_Multi_Interval_Division_Adjacent)

theory

Extended_Multi_Interval_Division_Adjacent

imports

Extended_Multi_Interval_Division_Core

begin

definition $\langle \text{minterval_adj_inverse } x = \text{mInterval_adj } (\text{mk_mInterval_adj } (\text{minverse } (\text{mlist_adj } x))) \rangle$

definition $\langle \text{minterval_adj_divide } x y = (\text{minterval_adj_inverse } y) * x \rangle$

lemma *set_of_adj_non_zero_map_inverse:*

assumes $\langle 0 \notin \text{set_of_adj } xs \rangle$

shows $\langle \text{concat } (\text{map } \text{inverse_interval } (\text{mlist_adj } xs)) = \text{map } (\lambda i. \text{mk_interval } (1 / \text{upper } i, 1 / \text{lower } i)) (\text{mlist_adj } xs) \rangle$

<proof>

interpretation *minterval_adj_division_inverse minterval_adj_divide minterval_adj_inverse*

<proof>

end

13.3 Bringing Everything Together (Extended_Multi_Interval_Division)

theory

Extended_Multi_Interval_Division

imports

Extended_Multi_Interval_Division_Overlapping

Extended_Multi_Interval_Division_Non_Overlapping
Extended_Multi_Interval_Division_Adjacent

begin

end

14 Extended Multi-Interval Analysis

(Extended_Multi_Interval_Analysis)

theory

Extended_Multi_Interval_Analysis

imports

Extended_Multi_Interval_Division

begin

This theory provides extended multi-interval analysis over the type extended reals. All operations work over multi-intervals, i.e., lists of (closed) intervals.

end

Bibliography

- [1] A. D. Brucker, T. Cameron-Burke, and A. Stell. Formally verified interval arithmetic and its application to program verification. In *13th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormalISE 2024)*. IEEE, 2024.
- [2] A. D. Brucker and A. Stell. Verifying feedforward neural networks for classification in Isabelle/HOL. In M. Chechik, J.-P. Katoen, and M. Leucker, editors, *Formal Methods (FM 2023)*. Lübeck, Germany, 2023. ISBN: 978-3-642-38915-3. URL: <http://www.brucker.ch/bibliography/abstract/brucker.ea-feedforward-nn-verification-2023>.
- [3] A. Harapanahalli, S. Jafarpour, and S. Coogan. A toolbox for fast interval arithmetic in numpy with an application to formal verification of neural network controlled systems. *CoRR*, abs/2306.15340, 2023. DOI: 10.48550/ARXIV.2306.15340. arXiv: 2306.15340. URL: <https://doi.org/10.48550/arXiv.2306.15340>.
- [4] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, USA, 2009. ISBN: 0898716691.
- [5] D. Ratz. *Inclusion isotone extended interval arithmetic. A toolbox update*. 1997. DOI: 10.5445/IR/67997. Karlsruhe 1996. (Berichte aus dem Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation. 1996,5.)
- [6] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal security analysis of neural networks using symbolic intervals. In *Proceedings of the 27th USENIX Conference on Security Symposium, SEC'18*, pages 1599–1614, Baltimore, MD, USA. USENIX Association, 2018. ISBN: 9781931971461.