Alpha-Beta Pruning

Tobias Nipkow
Technical University of Munich

March 17, 2025

Abstract

Alpha-beta pruning is an efficient search strategy for two-player game trees. It was invented in the late 1950s and is at the heart of most implementations of combinatorial game playing programs. These theories formalize and verify a number of variations of alpha-beta pruning, in particular fail-hard and fail-soft, and valuations into linear orders, distributive lattices and domains with negative values.

A detailed presentation of these theories can be found in the chapter *Alpha-Beta Pruning* in the (forthcoming) book Functional Data Structures and Algorithms — A Proof Assistant Approach.

Chapter 1

Overview

1.1 Introduction

Alpha-beta pruning is an efficient search strategy for two-player game trees. It was invented in the late 1950s and is at the heart of most implementations of combinatorial game playing programs. Most publications assume that the game values are numbers augmented with $\pm\infty$. This generalizes easily to an arbitrary linear order with $\bot$ and $\top$ values. We consider this standard case first. Later it was realized that alpha-beta pruning can be generalized to distributive lattices. We consider this case separately. In both cases we analyze two variants: *fail-hard* (analyzed by Knuth and Moore [3]) and *fail-soft* (introduced by Fishburn [2]). Our analysis focusses on functional correctness, not quantitative results. All algorithms operate on game trees of this type:

$$\textbf{datatype}\ 'a\ tree = Lf\ 'a \mid Nd\ ('a\ tree\ list)$$

1.2 Linear Orders

We assume that the type of values is a bounded linear order with $\bot$ and $\top$. Game trees are evaluated in the standard manner via functions *maxmin* (the maximizer) and the dual *minmax* (the minimizer).

$maxmin :: 'a\ tree \Rightarrow 'a$

$maxmin\ (Lf\ x) = x$
$maxmin\ (Nd\ ts) = maxs\ (map\ minmax\ ts)$

$minmax :: 'a\ tree \Rightarrow 'a$

$minmax\ (Lf\ x) = x$
$minmax\ (Nd\ ts) = mins\ (map\ maxmin\ ts)$

$maxs :: 'a\ list \Rightarrow 'a$

$$maxs\ [] = \bot$$
$$maxs\ (x\ \#\ xs) = max\ x\ (maxs\ xs)$$

$$mins :: 'a\ list \Rightarrow 'a$$

$$mins\ [] = \top$$
$$mins\ (x\ \#\ xs) = min\ x\ (mins\ xs)$$

The maximizer and minimizer functions are dual to each other. In this overview we will only show the maximizer side from now on.

1.2.1 Fail-Hard

The fail-hard variant of alpha-beta pruning is defined like this:

$$ab\_max :: 'a \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$$

$$ab\_max\ \_\ \_\ (Lf\ x) = x$$
$$ab\_max\ a\ b\ (Nd\ ts) = ab\_maxs\ a\ b\ ts$$

$$ab\_maxs :: 'a \Rightarrow 'a \Rightarrow 'a\ tree\ list \Rightarrow 'a$$

$$ab\_maxs\ a\ \_\ [] = a$$
$$ab\_maxs\ a\ b\ (t\ \#\ ts)$$
$$= (\mathbf{let}\ a' = max\ a\ (ab\_min\ a\ b\ t)$$
$$\quad \mathbf{in\ if}\ b \le a'\ \mathbf{then}\ a'\ \mathbf{else}\ ab\_maxs\ a'\ b\ ts)$$

The intuitive meaning of $ab\_max\ a\ b\ t$ roughly is this: search t, focussing on values in the interval from a to b. That is, a is the maximum value that the maximizer is already assured of and b is the minimum value that the minimizer is already assured of (by the search so far). During the search, the maximizer will increase a, the minimizer will decrease b.

The desired correctness property is that alpha-beta pruning with the full interval yields the value of the game tree:

$$ab\_max\ \bot\ \top\ t = maxmin\ t \tag{1.1}$$

Knuth and Moore generalize this property for arbitrary a and b, using the following predicate:

$$x \cong y\ (\mathrm{mod}\ a,b) \equiv$$
$$((y \le a \longrightarrow x \le a) \wedge$$
$$(a < y \wedge y < b \longrightarrow y = x) \wedge$$
$$(b \le y \longrightarrow b \le x))$$

It follows easily that $x \cong y\ (\mathrm{mod}\ \bot,\top)$ implies $x = y$. (Also interesting to note is commutativity: $a < b \Longrightarrow x \cong y\ (\mathrm{mod}\ a,b) = y \cong x\ (\mathrm{mod}\ a,b)$.) We have verified Knuth and Moore's correctness theorem

$$a < b \Longrightarrow maxmin\ t \cong ab\_max\ a\ b\ t\ (\mathrm{mod}\ a,b)$$

which immediately implies (1.1).

1.2.2 Fail-Soft

Fishburn introduced the fail-soft variant that agrees with fail-hard if the value is in between a and b but is more precise otherwise, where fail-hard just returns a or b:

$$ab\_max' :: 'a \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$$

$$ab\_max'\ \_\ \_\ (Lf\ x) = x$$
$$ab\_max'\ a\ b\ (Nd\ ts) = ab\_maxs'\ a\ b\ \bot\ ts$$

$$ab\_maxs' :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ tree\ list \Rightarrow 'a$$

$$ab\_maxs'\ \_\ \_\ m\ [] = m$$
$$ab\_maxs'\ a\ b\ m\ (t\ \#\ ts)$$
$$= (\textbf{let}\ m' = max\ m\ (ab\_min'\ (max\ m\ a)\ b\ t)$$
$$\quad \textbf{in if}\ b \le m'\ \textbf{then}\ m'\ \textbf{else}\ ab\_maxs'\ a\ b\ m'\ ts)$$

Fishburn claims that fail-soft searches the same part of the tree as fail-hard but that sometimes fail-soft bounds the real value more tightly than fail-hard because fail-soft satisfies

$$a < b \Longrightarrow ab\_max'\ a\ b\ t \le maxmin\ t\ (\mathrm{mod}\ a{,}b) \tag{1.2}$$

where $\le$ is a strengthened version of $\cong$:

$$ab \le v\ (\mathrm{mod}\ a{,}b) \equiv$$
$$((ab \le a \longrightarrow v \le ab) \wedge$$
$$(a < ab \wedge ab < b \longrightarrow ab = v) \wedge$$
$$(b \le ab \longrightarrow ab \le v))$$

We can confirm both claims. However, what Fishburn misses is that fail-hard already satisfies *fishburn*:

$$a < b \Longrightarrow ab\_max\ a\ b\ t \le maxmin\ t\ (\mathrm{mod}\ a{,}b)$$

Thus (1.2) does not imply that fail-soft is better. However, we have proved

$$a < b \Longrightarrow ab\_max\ a\ b\ t \le ab\_max'\ a\ b\ t\ (\mathrm{mod}\ a{,}b)$$

which does indeed show that fail-soft approximates the real value at least as well as fail-hard.

Fail-soft does not improve matters if one only looks at the top-level call with $\bot$ and $\top$. It comes into its own when the tree is actually a graph and nodes are visited repeatedly from different directions with different a and b which are typically not $\bot$ and $\top$. Then it becomes crucial to store the results of previous alpha-beta calls in a cache and use it to (possibly) narrow the search window on successive searches of the same subgraph. The justification: Let ab be some search function that *fishburn* the real value. If on a previous call $b \le ab\ a\ b\ t$, then in a subsequent search of the same t with possibly different a' and b', we can replace a' by $max\ a'\ (ab\ a\ b\ t)$:

$$[\![\forall a\ b.\ abf\ a\ b\ t \le maxmin\ t\ (\mathrm{mod}\ a,b);\ b \le abf\ a\ b\ t;\ max\ a'\ (abf\ a\ b\ t) < b']\!] \Longrightarrow abf\ (max\ a'\ (abf\ a\ b\ t))\ b'\ t \le maxmin\ t\ (\mathrm{mod}\ a',b')$$

There is a dual lemma for replacing b' by $min\ b'\ (ab\ a\ b\ t)$.

We have a verified version of alpha-beta pruning with a cache, but it is not yet part of this development.

1.2.3 Negation

Traditionally the definition of both the evaluation and of alpha-beta pruning does not define a minimizer and maximizer separately. Instead it defines only one function and uses negation (on numbers!) to flip between the two players. This is evaluation and the fail-hard and fail-soft variants of alpha-beta pruning:

$negmax :: {}'a\ tree \Rightarrow {}'a$

$negmax\ (Lf\ x) = x$
$negmax\ (Nd\ ts) = maxs\ (map\ (\lambda t.\ -\ negmax\ t)\ ts)$

$ab\_negmax :: {}'a \Rightarrow {}'a \Rightarrow {}'a\ tree \Rightarrow {}'a$

$ab\_negmax\ \_\ \_\ (Lf\ x) = x$
$ab\_negmax\ a\ b\ (Nd\ ts) = ab\_negmaxs\ a\ b\ ts$

$ab\_negmaxs :: {}'a \Rightarrow {}'a \Rightarrow {}'a\ tree\ list \Rightarrow {}'a$

$ab\_negmaxs\ a\ \_\ [] = a$
$ab\_negmaxs\ a\ b\ (t\ \#\ ts)$
$= (\mathbf{let}\ a' = max\ a\ (-\ ab\_negmax\ (-\ b)\ (-\ a)\ t)$
$\quad \mathbf{in\ if}\ b \le a'\ \mathbf{then}\ a'\ \mathbf{else}\ ab\_negmaxs\ a'\ b\ ts)$

$ab\_negmax' :: {}'a \Rightarrow {}'a \Rightarrow {}'a\ tree \Rightarrow {}'a$

$ab\_negmax'\ \_\ \_\ (Lf\ x) = x$
$ab\_negmax'\ a\ b\ (Nd\ ts) = ab\_negmaxs'\ a\ b\ \bot\ ts$

$ab\_negmaxs' :: {}'a \Rightarrow {}'a \Rightarrow {}'a \Rightarrow {}'a\ tree\ list \Rightarrow {}'a$

$ab\_negmaxs'\ \_\ \_\ m\ [] = m$
$ab\_negmaxs'\ a\ b\ m\ (t\ \#\ ts)$
$= (\mathbf{let}\ m' = max\ m\ (-\ ab\_negmax'\ (-\ b)\ (-\ max\ m\ a)\ t)$
$\quad \mathbf{in\ if}\ b \le m'\ \mathbf{then}\ m'\ \mathbf{else}\ ab\_negmaxs'\ a\ b\ m'\ ts)$

However, what does "$-$" on a linear order mean? It turns out that the following two properties of "$-$" are required to make things work:

$$-\ min\ x\ y = max\ (-\ x)\ (-\ y) \qquad -\ (-\ x) = x$$

We call such linear orders *de Morgan orders*. We have proved correctness of alpha-beta pruning on de Morgan orders:

$a < b \Longrightarrow ab\_negmax\ a\ b\ t \leq negmax\ t\ (\mathrm{mod}\ a{,}b)$
$a < b \Longrightarrow ab\_negmax'\ a\ b\ t \leq negmax\ t\ (\mathrm{mod}\ a{,}b)$
$a < b \Longrightarrow ab\_negmax\ a\ b\ t \leq ab\_negmax'\ a\ b\ t\ (\mathrm{mod}\ a{,}b)$

1.3 Lattices

Bird and Hughes [1] were the first to generalize alpha-beta pruning from linear orders to lattices. The generalization of the code is easy: simply replace *min* and *max* by ($\sqcap$) and ($\sqcup$). Thus, the value of a game tree is now defined like this:

$supinf :: 'a\ tree \Rightarrow 'a$

$supinf\ (Lf\ x) = x$
$supinf\ (Nd\ ts) = sups\ (map\ infsup\ ts)$

$sups :: 'a\ list \Rightarrow 'a$

$sups\ [] = \bot$
$sups\ (x\ \#\ xs) = x \sqcup sups\ xs$

And similarly fail-hard and fail-soft alpha-beta pruning:

$ab\_sup :: 'a \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$

$ab\_sup\ \_\ \_\ (Lf\ x) = x$
$ab\_sup\ a\ b\ (Nd\ ts) = ab\_sups\ a\ b\ ts$

$ab\_sups :: 'a \Rightarrow 'a \Rightarrow 'a\ tree\ list \Rightarrow 'a$

$ab\_sups\ a\ \_\ [] = a$
$ab\_sups\ a\ b\ (t\ \#\ ts)$
$= (\mathbf{let}\ a' = a \sqcup ab\_inf\ a\ b\ t$
$\quad \mathbf{in\ if}\ b \leq a'\ \mathbf{then}\ a'\ \mathbf{else}\ ab\_sups\ a'\ b\ ts)$

$ab\_sup' :: 'a \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$

$ab\_sup'\ \_\ \_\ (Lf\ x) = x$
$ab\_sup'\ a\ b\ (Nd\ ts) = ab\_sups'\ a\ b\ \bot\ ts$

$ab\_sups' :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ tree\ list \Rightarrow 'a$

$ab\_sups'\ \_\ \_\ m\ [] = m$
$ab\_sups'\ a\ b\ m\ (t\ \#\ ts)$
$= (\mathbf{let}\ m' = m \sqcup ab\_inf'\ (m \sqcup a)\ b\ t$
$\quad \mathbf{in\ if}\ b \leq m'\ \mathbf{then}\ m'\ \mathbf{else}\ ab\_sups'\ a\ b\ m'\ ts)$

It turns out that this version of alpha-beta pruning works for bounded distributive lattices. To prove this we can generalize both $\cong$ and $\leq$ by first rephrasing them (for linear orders)

$$a < b \Longrightarrow x \cong y \ (\mathrm{mod}\ a,b) = (\mathit{max}\ a\ (\mathit{min}\ x\ b) = \mathit{max}\ a\ (\mathit{min}\ y\ b))$$
$$a < b \Longrightarrow ab \leq v \ (\mathrm{mod}\ a,b) = (\mathit{min}\ v\ b \leq ab \wedge ab \leq \mathit{max}\ v\ a)$$

and then deriving from the right-hand sides new versions $\simeq$ and $\sqsubseteq$ appropriate for lattices:

$$x \simeq y \ (\mathrm{mod}\ a,b) \equiv a \sqcup x \sqcap b = a \sqcup y \sqcap b$$
$$ab \sqsubseteq v \ (\mathrm{mod}\ a,b) \equiv b \sqcap v \leq ab \wedge ab \leq a \sqcup v$$

As for linear orders, $\sqsubseteq$ implies $\simeq$, but not the other way around:

$$ab \sqsubseteq v \ (\mathrm{mod}\ a,b) \Longrightarrow ab \simeq v \ (\mathrm{mod}\ a,b)$$

Both fail-hard and fail-soft are correct w.r.t. $\sqsubseteq$:

$$\mathit{ab\_sup}\ a\ b\ t \sqsubseteq \mathit{supinf}\ t \ (\mathrm{mod}\ a,b)$$
$$\mathit{ab\_sup'}\ a\ b\ t \sqsubseteq \mathit{supinf}\ t \ (\mathrm{mod}\ a,b)$$

1.3.1 Negation

This time we extend bounded distributive lattices to *de Morgan algebras* by adding "$-$" and assuming

$$-\ (x \sqcap y) = -\ x \sqcup -\ y \qquad -\ (-\ x) = x$$

Game tree evaluation:

$$\mathit{negsup} :: {'a}\ \mathit{tree} \Rightarrow {'a}$$
$$\mathit{negsup}\ (\mathit{Lf}\ x) = x$$
$$\mathit{negsup}\ (\mathit{Nd}\ \mathit{ts}) = \mathit{sups}\ (\mathit{map}\ (\lambda t.\ -\ \mathit{negsup}\ t)\ \mathit{ts})$$

Fail-hard alpha-beta pruning:

$$\mathit{ab\_negsup} :: {'a} \Rightarrow {'a} \Rightarrow {'a}\ \mathit{tree} \Rightarrow {'a}$$
$$\mathit{ab\_negsup}\ \_\ \_\ (\mathit{Lf}\ x) = x$$
$$\mathit{ab\_negsup}\ a\ b\ (\mathit{Nd}\ \mathit{ts}) = \mathit{ab\_negsups}\ a\ b\ \mathit{ts}$$

$$\mathit{ab\_negsups} :: {'a} \Rightarrow {'a} \Rightarrow {'a}\ \mathit{tree}\ \mathit{list} \Rightarrow {'a}$$
$$\mathit{ab\_negsups}\ a\ \_\ [] = a$$
$$\mathit{ab\_negsups}\ a\ b\ (t\ \#\ \mathit{ts})$$
$$= (\mathbf{let}\ a' = a \sqcup -\ \mathit{ab\_negsup}\ (-\ b)\ (-\ a)\ t$$
$$\quad \mathbf{in\ if}\ b \leq a'\ \mathbf{then}\ a'\ \mathbf{else}\ \mathit{ab\_negsups}\ a'\ b\ \mathit{ts})$$

Fail-soft alpha-beta pruning:

$$\mathit{ab\_negsup'} :: {'a} \Rightarrow {'a} \Rightarrow {'a}\ \mathit{tree} \Rightarrow {'a}$$
$$\mathit{ab\_negsup'}\ \_\ \_\ (\mathit{Lf}\ x) = x$$
$$\mathit{ab\_negsup'}\ a\ b\ (\mathit{Nd}\ \mathit{ts}) = \mathit{ab\_negsups'}\ a\ b\ \bot\ \mathit{ts}$$

$$
\begin{aligned}
&ab\_negsups' :: {}'a \Rightarrow {}'a \Rightarrow {}'a \Rightarrow {}'a\ tree\ list \Rightarrow {}'a\\
&ab\_negsups'\ \_\ \_\ m\ [] = m\\
&ab\_negsups'\ a\ b\ m\ (t \# ts)\\
&= (\mathbf{let}\ m' = m \sqcup -\ ab\_negsup'\ (-\ b)\ (-\ (m \sqcup a))\ t\\
&\quad \mathbf{in\ if}\ b \le m'\ \mathbf{then}\ m'\ \mathbf{else}\ ab\_negsups'\ a\ b\ m'\ ts)
\end{aligned}
$$

Correctness:

$$
\begin{aligned}
&ab\_negsup\ a\ b\ t \sqsubseteq negsup\ t\ (\mathrm{mod}\ a{,}b)\\
&ab\_negsup'\ a\ b\ t \sqsubseteq negsup\ t\ (\mathrm{mod}\ a{,}b)
\end{aligned}
$$

Bibliography

[1] R. S. Bird and J. Hughes. The alpha-beta algorithm: An exercise in program transformation. *Inf. Process. Lett.*, 24(1):53–57, 1987.

[2] J. P. Fishburn. An optimization of alpha-beta search. *SIGART Newsl.*, 72:29–31, 1980.

[3] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artif. Intell.*, 6(4):293–326, 1975.

Contents

1 Overview **1**
1.1 Introduction . . . 1
1.2 Linear Orders . . . 1
1.2.1 Fail-Hard . . . 2
1.2.2 Fail-Soft . . . 3
1.2.3 Negation . . . 4
1.3 Lattices . . . 5
1.3.1 Negation . . . 6

2 Linear Orders **11**
2.1 Classes . . . 11
2.2 Game Tree Evaluation . . . 12
2.2.1 Parameterized by the orderings . . . 13
2.2.2 Negamax: de Morgan orders . . . 14
2.3 Specifications . . . 15
2.3.1 The squash operator *max a* (*min x b*) . . . 15
2.3.2 Fail-Hard and Soft . . . 15
2.4 Alpha-Beta for Linear Orders . . . 17
2.4.1 From the Left . . . 17
2.4.2 From the Right . . . 29
2.5 Alpha-Beta for De Morgan Orders . . . 36
2.5.1 From the Left, Fail-Hard . . . 36
2.5.2 From the Left, Fail-Soft . . . 38
2.5.3 From the Right, Fail-Hard . . . 40
2.5.4 From the Right, Fail-Soft . . . 43

3 Distributive Lattices **46**
3.1 Game Tree Evaluation . . . 46
3.2 Distributive Lattices . . . 46
3.2.1 Fail-Hard . . . 48
3.2.2 Fail-Soft . . . 52
3.3 De Morgan Algebras . . . 53
3.3.1 Fail-Hard . . . 55

3.3.2 Fail-Soft 55

4 An Application: Tic-Tac-Toe 57

Chapter 2

Linear Orders

theory *Alpha_Beta_Linear*
imports
  HOL−Library.Extended_Real
begin

2.1 Classes

notation
  bot (‹⊥›) **and**
  top (‹⊤›)

class *bounded_linorder* = *linorder* + *order_top* + *order_bot*
begin

lemma *bounded_linorder_collapse*:
assumes ¬ ⊥ < ⊤ **shows** (*x*::′*a*) = *y*
⟨*proof*⟩

end

class *de_morgan_order* = *bounded_linorder* + *uminus* +
assumes *de_morgan_min*: − *min x y* = *max* (− *x*) (− *y*)
assumes *neg_neg*[*simp*]: − (− *x*) = *x*
begin

lemma *de_morgan_max*: − *max x y* = *min* (− *x*) (− *y*)
⟨*proof*⟩

lemma *neg_top*[*simp*]: − ⊤ = ⊥
⟨*proof*⟩

lemma *neg_bot*[*simp*]: − ⊥ = ⊤
⟨*proof*⟩

lemma *uminus_eq_iff*[*simp*]: $-a = -b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *uminus_le_reorder*: $(- a \leq b) = (- b \leq a)$
⟨*proof*⟩

lemma *uminus_less_reorder*: $(- a < b) = (- b < a)$
⟨*proof*⟩

lemma *minus_le_minus*[*simp*]: $- a \leq - b \longleftrightarrow b \leq a$
⟨*proof*⟩

lemma *minus_less_minus*[*simp*]: $- a < - b \longleftrightarrow b < a$
⟨*proof*⟩

lemma *less_uminus_reorder*: $a < - b \longleftrightarrow b < - a$
⟨*proof*⟩

end

instance *bool* :: *bounded_linorder* ⟨*proof*⟩

instantiation *ereal* :: *de_morgan_order*
begin

instance
⟨*proof*⟩

end

2.2 Game Tree Evaluation

datatype *'a tree* = *Lf 'a* | *Nd 'a tree list*

datatype_compat *tree*

fun *maxs* :: (*'a*::*bounded_linorder*) *list* ⇒ *'a* **where**
maxs [] = ⊥ |
maxs (*x*#*xs*) = *max x* (*maxs xs*)

fun *mins* :: (*'a*::*bounded_linorder*) *list* ⇒ *'a***where**
mins [] = ⊤ |
mins (*x*#*xs*) = *min x* (*mins xs*)

fun *maxmin* :: (*'a*::*bounded_linorder*) *tree* ⇒ *'a*
and *minmax* :: (*'a*::*bounded_linorder*) *tree* ⇒ *'a* **where**
maxmin (*Lf x*) = *x* |

maxmin (*Nd ts*) = *maxs* (*map minmax ts*) |
minmax (*Lf x*) = *x* |
minmax (*Nd ts*) = *mins* (*map maxmin ts*)

Cannot use *Max* instead of *maxs* because *Max* {} is undefined.

No need for bounds if lists are nonempty:

fun *invar* :: *'a tree* ⇒ *bool* **where**
invar (*Lf x*) = *True* |
invar (*Nd ts*) = (*ts* ≠ [] ∧ (∀ *t* ∈ *set ts*. *invar t*))

fun *maxs*1 :: (*'a*::*linorder*) *list* ⇒ *'a* **where**
*maxs*1 [*x*] = *x* |
*maxs*1 (*x*#*xs*) = *max x* (*maxs*1 *xs*)

fun *mins*1 :: (*'a*::*linorder*) *list* ⇒ *'a***where**
*mins*1 [*x*] = *x* |
*mins*1 (*x*#*xs*) = *min x* (*mins*1 *xs*)

fun *maxmin*1 :: (*'a*::*bounded_linorder*) *tree* ⇒ *'a*
and *minmax*1 :: (*'a*::*bounded_linorder*) *tree* ⇒ *'a* **where**
*maxmin*1 (*Lf x*) = *x* |
*maxmin*1 (*Nd ts*) = *maxs*1 (*map minmax*1 *ts*) |
*minmax*1 (*Lf x*) = *x* |
*minmax*1 (*Nd ts*) = *mins*1 (*map maxmin*1 *ts*)

lemma *maxs1_maxs*: *xs* ≠ [] ⟹ *maxs*1 *xs* = *maxs xs*
⟨*proof*⟩

lemma *mins1_mins*: *xs* ≠ [] ⟹ *mins*1 *xs* = *mins xs*
⟨*proof*⟩

lemma *maxmin1_maxmin*:
shows *invar t* ⟹ *maxmin*1 *t* = *maxmin t*
and *invar t* ⟹ *minmax*1 *t* = *minmax t*
⟨*proof*⟩

2.2.1 Parameterized by the orderings

fun *maxs_le* :: *'a* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a list* ⇒ *'a* **where**
maxs_le bo le [] = *bo* |
maxs_le bo le (*x*#*xs*) = (*let m* = *maxs_le bo le xs in if le x m then m else x*)

fun *maxmin_le* :: *'a* ⇒ *'a* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *'a tree* ⇒ *'a* **where**
maxmin_le _ _ _ (*Lf x*) = *x* |
maxmin_le bo to le (*Nd ts*) = *maxs_le bo le* (*map* (*maxmin_le to bo* (λ*x y*. *le y x*)) *ts*)

lemma *maxs_le_maxs*: *maxs_le* ⊥ (≤) *xs* = *maxs xs*
⟨*proof*⟩

lemma $maxs\_le\_mins$: $maxs\_le\ \top\ (\geq)\ xs = mins\ xs$
⟨*proof*⟩

lemma $maxmin\_le\_maxmin$:
shows $maxmin\_le\ \bot\ \top\ (\leq)\ t = maxmin\ \ t$
and $maxmin\_le\ \top\ \bot\ (\geq)\ t = minmax\ t$
⟨*proof*⟩

2.2.2 Negamax: de Morgan orders

fun $negmax :: ('a::de\_morgan\_order)\ tree \Rightarrow 'a$ **where**
$negmax\ (Lf\ x) = x\ |$
$negmax\ (Nd\ ts) = maxs\ (map\ (\lambda t.\ -\ negmax\ t)\ ts)$

lemma de_morgan_mins:
fixes $f :: 'a \Rightarrow 'b::de\_morgan\_order$
shows $-\ mins\ (map\ f\ xs) = maxs\ (map\ (\lambda x.\ -\ f\ x)\ xs)$
⟨*proof*⟩

fun $negate :: bool \Rightarrow ('a::de\_morgan\_order)\ tree \Rightarrow ('a::de\_morgan\_order)\ tree$
where
$negate\ b\ (Lf\ x) = Lf\ (if\ b\ then\ -x\ else\ x)\ |$
$negate\ b\ (Nd\ ts) = Nd\ (map\ (negate\ (\neg b))\ ts)$

lemma $negate\_negate$: $negate\ f\ (negate\ f\ t) = t$
⟨*proof*⟩

lemma $maxmin\_negmax$: $maxmin\ t = negmax\ (negate\ False\ t)$
and $minmax\_negmax$: $minmax\ t = -\ negmax\ (negate\ True\ t)$
⟨*proof*⟩

lemma $maxmin\ t = negmax\ (negate\ False\ t)$
and $minmax\ t = -\ negmax\ (negate\ True\ t)$
⟨*proof*⟩

lemma shows $negmax\_maxmin$: $negmax\ t = maxmin(negate\ False\ t)$
and $negmax\ t = -\ minmax(negate\ True\ t)$
⟨*proof*⟩

lemma $maxs\_append$: $maxs\ (xs\ @\ ys) = max\ (maxs\ xs)\ (maxs\ ys)$
⟨*proof*⟩

lemma $maxs\_rev$: $maxs\ (rev\ xs) = maxs\ xs$
⟨*proof*⟩

2.3 Specifications

2.3.1 The squash operator $max\ a\ (min\ x\ b)$

abbreviation *mm* **where** $mm\ a\ x\ b == min\ (max\ a\ x)\ b$

lemma *max_min_commute*: $(a{::}\_{::}linorder) \le b \Longrightarrow max\ a\ (min\ x\ b) = min\ b\ (max\ x\ a)$
⟨*proof*⟩

lemma *max_min_commute2*: $(a{::}\_{::}linorder) \le b \Longrightarrow max\ a\ (min\ x\ b) = min\ (max\ a\ x)\ b$
⟨*proof*⟩

lemma *max_min_neg*: $a{<}b \Longrightarrow max\ (a{::}\_{::}de\_morgan\_order)\ (min\ x\ b) = -\ max\ (-b)\ (min\ (-x)\ (-a))$
⟨*proof*⟩

2.3.2 Fail-Hard and Soft

Specification of fail-hard; symmetric in x and y!

abbreviation
$knuth\ (a{::}\_{::}linorder)\ b\ x\ y ==$
$((y \le a \longrightarrow x \le a) \wedge (a < y \wedge y < b \longrightarrow y = x) \wedge (b \le y \longrightarrow b \le x))$

abbreviation *knuth2* :: ('*a*::*linorder*) ⇒ '*a* ⇒ '*a* ⇒ '*a* ⇒ *bool* (‹(_ ≅/ _/ '(mod _,_'))› [51,51,0,0])
where $knuth2\ x\ y\ a\ b \equiv knuth\ a\ b\ x\ y$

notation (*latex* **output**) *knuth2* (‹(_ ≅/ _/ '(mod _,_'))› [51,51,0,0])

lemma *knuth_bot_top*: $knuth\ \bot\ \top\ x\ y \Longrightarrow x = (y{::}\_{::}bounded\_linorder)$
⟨*proof*⟩

The equational version of *knuth*. First, automatically:

lemma *knuth_iff_max_min*: $a < b \Longrightarrow knuth\ a\ b\ x\ y \longleftrightarrow max\ a\ (min\ x\ b) = max\ a\ (min\ y\ b)$
⟨*proof*⟩

Needs $a < b$: take everything $= \infty$, x = 0

lemma *knuth_if_mm*: $a < b \Longrightarrow mm\ a\ y\ b = mm\ a\ x\ b \Longrightarrow knuth\ a\ b\ x\ y$
⟨*proof*⟩

lemma *mm_if_knuth*: $knuth\ a\ b\ y\ x \Longrightarrow mm\ a\ y\ b = mm\ a\ x\ b$
⟨*proof*⟩

Now readable:

lemma *mm_iff_knuth*: **assumes** $(a{::}\_{::}linorder) < b$
shows $max\ a\ (min\ x\ b) = max\ a\ (min\ y\ b) \longleftrightarrow knuth\ a\ b\ y\ x$ (**is** *?mm* = *?h*)

⟨*proof*⟩

corollary *mm_iff_knuth′*: $a < b \Longrightarrow max\ a\ (min\ x\ b) = max\ a\ (min\ y\ b) \longleftrightarrow knuth\ a\ b\ x\ y$
⟨*proof*⟩

corollary *knuth_comm*: $a<b \Longrightarrow knuth\ a\ b\ x\ y \longleftrightarrow knuth\ a\ b\ y\ x$
⟨*proof*⟩

Specification of fail-soft: v is the actual value, ab the approximation.

abbreviation
fishburn (*a*::_::*linorder*) *b v ab* ==
$((ab \le a \longrightarrow v \le ab) \wedge (a < ab \wedge ab < b \longrightarrow ab = v) \wedge (b \le ab \longrightarrow ab \le v))$

abbreviation *fishburn2* :: (′*a*::*linorder*) ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ *bool* (‹(_ ≤/ _/ ′(*mod* _,_′))› [51,51,0,0])
where *fishburn2 ab v a b* ≡ *fishburn a b v ab*

notation (*latex* **output**) *fishburn2* (‹(_ ≤/ _/ ′(mod _,_′))› [51,51,0,0])

lemma *fishburn_iff_min_max*: $a < b \Longrightarrow fishburn\ a\ b\ v\ ab \longleftrightarrow min\ v\ b \le ab \wedge ab \le max\ v\ a$
⟨*proof*⟩

lemma *knuth_if_fishburn*: $fishburn\ a\ b\ x\ y \Longrightarrow knuth\ a\ b\ x\ y$
⟨*proof*⟩

corollary *fishburn_bot_top*: *fishburn* ⊥ ⊤ (*x*::_::*bounded_linorder*) $y \Longrightarrow x = y$
⟨*proof*⟩

lemma *trans_fishburn*: $fishburn\ a\ b\ x\ y \Longrightarrow fishburn\ a\ b\ y\ z \Longrightarrow fishburn\ a\ b\ x\ z$
⟨*proof*⟩

An simple alternative formulation:

lemma *fishburn2*: $a < b \Longrightarrow fishburn\ a\ b\ f\ g = ((g > a \longrightarrow f \ge g) \wedge (g < b \longrightarrow f \le g))$
⟨*proof*⟩

Like *fishburn2* above, but exchanging f and g. Not clearly related to *knuth* and *fishburn*.

abbreviation *lb_ub a b f g* ≡ $((f \ge a \longrightarrow g \ge a) \wedge (f \le b \longrightarrow g \le b))$

lemma (*a*::*nat*) $< b \Longrightarrow knuth\ a\ b\ f\ g \Longrightarrow$ *lb_ub a b f g*
quickcheck[*expect*=*counterexample*]
⟨*proof*⟩

lemma (*a*::*nat*) $< b \Longrightarrow$ *lb_ub a b f g* $\Longrightarrow knuth\ a\ b\ f\ g$

quickcheck[*expect=counterexample*]
⟨*proof*⟩

lemma *fishburn a b f g* $\Longrightarrow$ *lb_ub a b f g*
⟨*proof*⟩

lemma (*a*::*nat*) $<$ *b* $\Longrightarrow$ *lb_ub a b f g* $\Longrightarrow$ *fishburn a b f g*
quickcheck[*expect=counterexample*]
⟨*proof*⟩

lemma *a*<(*b*::*int*) $\Longrightarrow$ *fishburn a b f g* $\Longrightarrow$ *fishburn a b g f*
quickcheck[*expect=counterexample*]
⟨*proof*⟩

lemma *a*<(*b*::*int*) $\Longrightarrow$ *knuth a b f g* $\Longrightarrow$ *fishburn a b f g*
quickcheck[*expect=counterexample*]
⟨*proof*⟩

lemma *fishburn_trans*: *fishburn a b f g* $\Longrightarrow$ *fishburn a b g h* $\Longrightarrow$ *fishburn a b f h*
⟨*proof*⟩

Exactness: if the real value is within the bounds, *ab* is exact. More interesting would be the other way around. The impact of the exactness lemmas below is unclear.

lemma *fishburn_exact*: $a \le v \wedge v \le b$ $\Longrightarrow$ *fishburn a b v ab* $\Longrightarrow$ $ab = v$
⟨*proof*⟩

Let everyting = 0 and $ab = 1$:

lemma *mm_not_exact*: $a \le$ (*v*::*bool*) $\wedge\ v \le b$ $\Longrightarrow$ *mm a v b* = *mm a ab b* $\Longrightarrow$ *ab* = *v*
quickcheck[*expect=counterexample*]
⟨*proof*⟩
lemma *knuth_not_exact*: $a \le$ (*v*::*ereal*) $\wedge\ v \le b$ $\Longrightarrow$ *knuth a b v ab* $\Longrightarrow$ *ab* = *v*
quickcheck[*expect=counterexample*]
⟨*proof*⟩
lemma *mm_not_exact*: $a < b$ $\Longrightarrow$ (*a*::*ereal*) $\le v \wedge v \le b$ $\Longrightarrow$ *mm a v b* = *mm a ab b* $\Longrightarrow$ *ab* = *v*
quickcheck[*expect=counterexample*]
⟨*proof*⟩

2.4 Alpha-Beta for Linear Orders

2.4.1 From the Left

Hard

fun *ab_max* :: *'a* ⇒ *'a* ⇒ (*'a*::*linorder*)*tree* ⇒ *'a* **and** *ab_maxs ab_min ab_mins* **where**
ab_max a b (*Lf x*) = *x* |

$ab\_max\ a\ b\ (Nd\ ts) = ab\_maxs\ a\ b\ ts$ |

$ab\_maxs\ a\ b\ [] = a$ |
$ab\_maxs\ a\ b\ (t\#ts) = (let\ a' = max\ a\ (ab\_min\ a\ b\ t)\ in\ if\ a' \geq b\ then\ a'\ else\ ab\_maxs\ a'\ b\ ts)$ |

$ab\_min\ a\ b\ (Lf\ x) = x$ |
$ab\_min\ a\ b\ (Nd\ ts) = ab\_mins\ a\ b\ ts$ |

$ab\_mins\ a\ b\ [] = b$ |
$ab\_mins\ a\ b\ (t\#ts) = (let\ b' = min\ b\ (ab\_max\ a\ b\ t)\ in\ if\ b' \leq a\ then\ b'\ else\ ab\_mins\ a\ b'\ ts)$

lemma *ab_maxs_ge_a*: $ab\_maxs\ a\ b\ ts \geq a$
⟨*proof*⟩

lemma *ab_mins_le_b*: $ab\_mins\ a\ b\ ts \leq b$
⟨*proof*⟩

Automatic *fishburn* proof:

theorem
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t)\ \ (ab\_max\ a\ b\ t)$
and $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ (Nd\ ts))\ (ab\_maxs\ a\ b\ ts)$
and $a < b \Longrightarrow fishburn\ a\ b\ (minmax\ t)\ \ (ab\_min\ a\ b\ t)$
and $a < b \Longrightarrow fishburn\ a\ b\ (minmax\ (Nd\ ts))\ (ab\_mins\ a\ b\ ts)$
⟨*proof*⟩

Detailed *fishburn* proof:

theorem *fishburn_val_ab*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t)\ \ (ab\_max\ a\ b\ t)$
and $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ (Nd\ ts))\ (ab\_maxs\ a\ b\ ts)$
and $a < b \Longrightarrow fishburn\ a\ b\ (minmax\ t)\ \ (ab\_min\ a\ b\ t)$
and $a < b \Longrightarrow fishburn\ a\ b\ (minmax\ (Nd\ ts))\ (ab\_mins\ a\ b\ ts)$
⟨*proof*⟩

corollary *ab_max_bot_top*: $ab\_max\ \bot\ \top\ t = maxmin\ t$
⟨*proof*⟩

A detailed *knuth* proof, similar to $a < b \Longrightarrow ab\_max\ a\ b\ t \leq maxmin\ t$ (mod a,b)

$a < b \Longrightarrow ab\_maxs\ a\ b\ ts \leq maxmin\ (Nd\ ts)$ (mod a,b)

$a < b \Longrightarrow ab\_min\ a\ b\ t \leq minmax\ t$ (mod a,b)

$a < b \Longrightarrow ab\_mins\ a\ b\ ts \leq minmax\ (Nd\ ts)$ (mod a,b) proof:

theorem *knuth_val_ab*:
shows $a < b \Longrightarrow knuth\ a\ b\ (maxmin\ t)\ \ (ab\_max\ a\ b\ t)$
and $a < b \Longrightarrow knuth\ a\ b\ (maxmin\ (Nd\ ts))\ (ab\_maxs\ a\ b\ ts)$
and $a < b \Longrightarrow knuth\ a\ b\ (minmax\ t)\ \ (ab\_min\ a\ b\ t)$
and $a < b \Longrightarrow knuth\ a\ b\ (minmax\ (Nd\ ts))\ (ab\_mins\ a\ b\ ts)$
⟨*proof*⟩

Towards exactness:

lemma $ab\_max\_le\_b$: $\llbracket\ a \le b;\ maxmin\ t \le b\ \rrbracket \Longrightarrow ab\_max\ a\ b\ t \le b$
and $\llbracket\ a \le b;\ maxmin\ (Nd\ ts) \le b\ \rrbracket \Longrightarrow ab\_maxs\ a\ b\ ts \le b$
and $\llbracket\ a \le minmax\ t;\ a \le b\ \rrbracket \Longrightarrow a \le ab\_min\ a\ b\ t$
and $\llbracket\ a \le minmax\ (Nd\ ts);\ a \le b\ \rrbracket \Longrightarrow a \le ab\_mins\ a\ b\ ts$
⟨*proof*⟩

lemma ab_max_exact:
assumes $v = maxmin\ t\ a \le v \wedge v \le b$
shows $ab\_max\ a\ b\ t = v$
⟨*proof*⟩

Hard, max/min flag

fun $ab\_minmax :: bool \Rightarrow ('a::linorder) \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **and** $ab\_minmaxs$
where
$ab\_minmax\ mx\ a\ b\ (Lf\ x) = x\ |$
$ab\_minmax\ mx\ a\ b\ (Nd\ ts) = ab\_minmaxs\ mx\ a\ b\ ts\ |$

$ab\_minmaxs\ mx\ a\ b\ [] = a\ |$
$ab\_minmaxs\ mx\ a\ b\ (t\#ts) =$
$(let\ abt = ab\_minmax\ (\neg mx)\ b\ a\ t;$
$a' = (if\ mx\ then\ max\ else\ min)\ a\ abt$
$in\ if\ (if\ mx\ then\ (\ge)\ else\ (\le))\ a'\ b\ then\ a'\ else\ ab\_minmaxs\ mx\ a'\ b\ ts)$

lemma $ab\_max\_ab\_minmax$:
shows $ab\_max\ a\ b\ t = ab\_minmax\ True\ a\ b\ t$
and $ab\_maxs\ a\ b\ ts = ab\_minmaxs\ True\ a\ b\ ts$
and $ab\_min\ b\ a\ t = ab\_minmax\ False\ a\ b\ t$
and $ab\_mins\ b\ a\ ts = ab\_minmaxs\ False\ a\ b\ ts$
⟨*proof*⟩

Hard, abstracted over ≤

fun $ab\_le :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a::linorder)tree \Rightarrow 'a$ **and** ab_les
where
$ab\_le\ le\ a\ b\ (Lf\ x) = x\ |$
$ab\_le\ le\ a\ b\ (Nd\ ts) = ab\_les\ le\ a\ b\ ts\ |$

$ab\_les\ le\ a\ b\ [] = a\ |$
$ab\_les\ le\ a\ b\ (t\#ts) = (let\ abt = ab\_le\ (\lambda x\ y.\ le\ y\ x)\ b\ a\ t;$
$a' = if\ le\ a\ abt\ then\ abt\ else\ a\ in\ if\ le\ b\ a'\ then\ a'\ else\ ab\_les\ le\ a'\ b\ ts)$

lemma $ab\_max\_ab\_le$:
shows $ab\_max\ a\ b\ t = ab\_le\ (\le)\ a\ b\ t$
and $ab\_maxs\ a\ b\ ts = ab\_les\ (\le)\ a\ b\ ts$
and $ab\_min\ b\ a\ t = ab\_le\ (\ge)\ a\ b\ t$
and $ab\_mins\ b\ a\ ts = ab\_les\ (\ge)\ a\ b\ ts$
⟨*proof*⟩

Delayed test:

fun $ab\_le3 :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a::linorder)tree \Rightarrow 'a$ **and** ab_les3 **where**
$ab\_le3\ le\ a\ b\ (Lf\ x) = x$ |
$ab\_le3\ le\ a\ b\ (Nd\ ts) = ab\_les3\ le\ a\ b\ ts$ |

$ab\_les3\ le\ a\ b\ [] = a$ |
$ab\_les3\ le\ a\ b\ (t\#ts) =$
$(if\ le\ b\ a\ then\ a\ else$
$let\ abt = ab\_le3\ (\lambda x\ y.\ le\ y\ x)\ b\ a\ t;$
$a' = if\ le\ a\ abt\ then\ abt\ else\ a$
$in\ ab\_les3\ le\ a'\ b\ ts)$

lemma $ab\_max\_ab\_le3$:
shows $a < b \Longrightarrow ab\_max\ a\ b\ t = ab\_le3\ (\le)\ a\ b\ t$
and $a < b \Longrightarrow ab\_maxs\ a\ b\ ts = ab\_les3\ (\le)\ a\ b\ ts$
and $a > b \Longrightarrow ab\_min\ b\ a\ t = ab\_le3\ (\ge)\ a\ b\ t$
and $a > b \Longrightarrow ab\_mins\ b\ a\ ts = ab\_les3\ (\ge)\ a\ b\ ts$
$\langle proof \rangle$

corollary $ab\_le3\_bot\_top$: $ab\_le3\ (\le)\ \bot\ \top\ t = maxmin\ t$
$\langle proof \rangle$

Hard, max/min in Lf

Idea due to Bird and Hughes

fun $ab\_max2 :: 'a \Rightarrow 'a \Rightarrow ('a::linorder)tree \Rightarrow 'a$ **and** ab_maxs2 **and** ab_min2 **and** ab_mins2 **where**
$ab\_max2\ a\ b\ (Lf\ x) = max\ a\ (min\ x\ b)$ |
$ab\_max2\ a\ b\ (Nd\ ts) = ab\_maxs2\ a\ b\ ts$ |

$ab\_maxs2\ a\ b\ [] = a$ |
$ab\_maxs2\ a\ b\ (t\#ts) = (let\ a' = ab\_min2\ a\ b\ t\ in\ if\ a' = b\ then\ a'\ else\ ab\_maxs2\ a'\ b\ ts)$ |

$ab\_min2\ a\ b\ (Lf\ x) = max\ a\ (min\ x\ b)$ |
$ab\_min2\ a\ b\ (Nd\ ts) = ab\_mins2\ a\ b\ ts$ |

$ab\_mins2\ a\ b\ [] = b$ |
$ab\_mins2\ a\ b\ (t\#ts) = (let\ b' = ab\_max2\ a\ b\ t\ in\ if\ a = b'\ then\ b'\ else\ ab\_mins2\ a\ b'\ ts)$

lemma $ab\_max2\_max\_min\_maxmin$:
shows $a \le b \Longrightarrow ab\_max2\ a\ b\ t = max\ a\ (min\ (maxmin\ t)\ b)$
and $a \le b \Longrightarrow ab\_maxs2\ a\ b\ ts = max\ a\ (min\ (maxmin\ (Nd\ ts))\ b)$
and $a \le b \Longrightarrow ab\_min2\ a\ b\ t = max\ a\ (min\ (minmax\ t)\ b)$
and $a \le b \Longrightarrow ab\_mins2\ a\ b\ ts = max\ a\ (min\ (minmax\ (Nd\ ts))\ b)$
$\langle proof \rangle$

corollary *ab_max2_bot_top*: $ab\_max2\ \bot\ \top\ t = maxmin\ t$
⟨*proof*⟩

Now for the *ab* version parameterized with *le*:

fun $ab\_le2 :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a::linorder)tree \Rightarrow 'a$ **and** *ab_les2* **where**
$ab\_le2\ le\ a\ b\ (Lf\ x) =$
$(let\ xb = if\ le\ x\ b\ then\ x\ else\ b$
$in\ if\ le\ a\ xb\ then\ xb\ else\ a)$ |
$ab\_le2\ le\ a\ b\ (Nd\ ts) = ab\_les2\ le\ a\ b\ ts$ |

$ab\_les2\ le\ a\ b\ [] = a$ |
$ab\_les2\ le\ a\ b\ (t\#ts) = (let\ a' = ab\_le2\ (\lambda x\ y.\ le\ y\ x)\ b\ a\ t\ in\ if\ a' = b\ then\ a'\ else\ ab\_les2\ le\ a'\ b\ ts)$

Relate *ab_le2* back to *ab_max2* (using $a \le b \Longrightarrow ab\_max2\ a\ b\ t = max\ a\ (min\ (maxmin\ t)\ b)$
$a \le b \Longrightarrow ab\_maxs2\ a\ b\ ts = max\ a\ (min\ (maxmin\ (Nd\ ts))\ b)$
$a \le b \Longrightarrow ab\_min2\ a\ b\ t = max\ a\ (min\ (minmax\ t)\ b)$
$a \le b \Longrightarrow ab\_mins2\ a\ b\ ts = max\ a\ (min\ (minmax\ (Nd\ ts))\ b)$!):

lemma *ab_le2_ab_max2*:
fixes $a :: \_ :: bounded\_linorder$
shows $a \le b \Longrightarrow ab\_le2\ (\le)\ a\ b\ t = ab\_max2\ a\ b\ t$
and $a \le b \Longrightarrow ab\_les2\ (\le)\ a\ b\ ts = ab\_maxs2\ a\ b\ ts$
and $a \le b \Longrightarrow ab\_le2\ (\ge)\ b\ a\ t = ab\_min2\ a\ b\ t$
and $a \le b \Longrightarrow ab\_les2\ (\ge)\ b\ a\ ts = ab\_mins2\ a\ b\ ts$
⟨*proof*⟩

corollary *ab_le2_bot_top*: $ab\_le2\ (\le)\ \bot\ \top\ t = maxmin\ t$
⟨*proof*⟩

Hard, Delayed Test

fun $ab\_max3 :: 'a \Rightarrow 'a \Rightarrow ('a::linorder)tree \Rightarrow 'a$ **and** *ab_maxs3* **and** *ab_min3* **and** *ab_mins3* **where**
$ab\_max3\ a\ b\ (Lf\ x) = x$ |
$ab\_max3\ a\ b\ (Nd\ ts) = ab\_maxs3\ a\ b\ ts$ |

$ab\_maxs3\ a\ b\ [] = a$ |
$ab\_maxs3\ a\ b\ (t\#ts) = (if\ a \ge b\ then\ a\ else\ ab\_maxs3\ (max\ a\ (ab\_min3\ a\ b\ t))\ b\ ts)$ |

$ab\_min3\ a\ b\ (Lf\ x) = x$ |
$ab\_min3\ a\ b\ (Nd\ ts) = ab\_mins3\ a\ b\ ts$ |

$ab\_mins3\ a\ b\ [] = b$ |
$ab\_mins3\ a\ b\ (t\#ts) = (if\ a \ge b\ then\ b\ else\ ab\_mins3\ a\ (min\ b\ (ab\_max3\ a\ b\ t))\ ts)$

lemma *ab_max3_ab_max*:
shows $a<b \Longrightarrow ab\_max3\ a\ b\ t = ab\_max\ a\ b\ t$
and $a<b \Longrightarrow ab\_maxs3\ a\ b\ ts = ab\_maxs\ a\ b\ ts$
and $a<b \Longrightarrow ab\_min3\ a\ b\ t = ab\_min\ a\ b\ t$
and $a<b \Longrightarrow ab\_mins3\ a\ b\ ts = ab\_mins\ a\ b\ ts$
⟨*proof*⟩

corollary *ab_max3_bot_top*: $ab\_max3\ \bot\ \top\ t = maxmin\ t$
⟨*proof*⟩

Soft

Fishburn

fun ab_max' :: $'a::bounded\_linorder \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **and** ab_maxs' ab_min' ab_mins' **where**
$ab\_max'\ a\ b\ (Lf\ x) = x$ |
$ab\_max'\ a\ b\ (Nd\ ts) = ab\_maxs'\ a\ b\ \bot\ ts$ |

$ab\_maxs'\ a\ b\ m\ [] = m$ |
$ab\_maxs'\ a\ b\ m\ (t\#ts) =$
$(let\ m' = max\ m\ (ab\_min'\ (max\ m\ a)\ b\ t)\ in\ if\ m' \geq b\ then\ m'\ else\ ab\_maxs'\ a\ b\ m'\ ts)$ |

$ab\_min'\ a\ b\ (Lf\ x) = x$ |
$ab\_min'\ a\ b\ (Nd\ ts) = ab\_mins'\ a\ b\ \top\ ts$ |

$ab\_mins'\ a\ b\ m\ [] = m$ |
$ab\_mins'\ a\ b\ m\ (t\#ts) =$
$(let\ m' = min\ m\ (ab\_max'\ a\ (min\ m\ b)\ t)\ in\ if\ m' \leq a\ then\ m'\ else\ ab\_mins'\ a\ b\ m'\ ts)$

lemma *ab_maxs'_ge_a*: $ab\_maxs'\ a\ b\ m\ ts \geq m$
⟨*proof*⟩

lemma *ab_mins'_le_a*: $ab\_mins'\ a\ b\ m\ ts \leq m$
⟨*proof*⟩

Find a, b and t such that $a < b$ and fail-soft is closer to the real value than fail-hard.

lemma $let\ a = -\infty;\ b = ereal\ 0;\ t = Nd\ [Nd\ []]$
$in\ a < b \wedge ab\_max\ a\ b\ t = 0 \wedge ab\_max'\ a\ b\ t = \infty \wedge maxmin\ t = \infty$
⟨*proof*⟩

theorem *fishburn_val_ab'*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t)\ (ab\_max'\ a\ b\ t)$

and *max m a* < *b* ⟹ *fishburn* (*max m a*) *b* (*maxmin* (*Nd ts*)) (*ab_maxs' a b m ts*)
and *a* < *b* ⟹ *fishburn a b* (*minmax t*) (*ab_min' a b t*)
and *a* < *min m b* ⟹ *fishburn a* (*min m b*) (*minmax* (*Nd ts*)) (*ab_mins' a b m ts*)
⟨*proof*⟩

theorem *fishburn_ab'_ab*:
shows *a* < *b* ⟹ *fishburn a b* (*ab_max' a b t*) (*ab_max a b t*)
and *max m a* < *b* ⟹ *fishburn a b* (*ab_maxs' a b m ts*) (*ab_maxs* (*max m a*) *b ts*)
and *a* < *b* ⟹ *fishburn a b* (*ab_min' a b t*) (*ab_min a b t*)
and *a* < *min m b* ⟹ *a* < *m* ⟹ *fishburn a b* (*ab_mins' a b m ts*) (*ab_mins a* (*min m b*) *ts*)
⟨*proof*⟩

Fail-soft can be more precise than fail-hard:

lemma *let a* = *ereal* 0; *b* = 1; *t* = *Nd* [] *in*
maxmin t = *ab_max' a b t* ∧ *maxmin t* ≠ *ab_max a b t*
⟨*proof*⟩

lemma *ab_max'_lb_ub*:
shows *a* ≤ *b* ⟹ *lb_ub a b* (*maxmin t*) (*ab_max' a b t*)
and *a* ≤ *b* ⟹ *lb_ub a b* (*max i* (*maxmin* (*Nd ts*))) (*ab_maxs' a b i ts*)
and *a* ≤ *b* ⟹ *lb_ub a b* (*minmax t*) (*ab_min' a b t*)
and *a* ≤ *b* ⟹ *lb_ub a b* (*min i* (*minmax* (*Nd ts*))) (*ab_mins' a b i ts*)
⟨*proof*⟩

lemma *ab_max'_exact_less*: ⟦ *a*<*b*; *v* = *maxmin t*; *a* ≤ *v* ∧ *v* ≤ *b* ⟧ ⟹ *ab_max' a b t* = *v*
⟨*proof*⟩

lemma *ab_max'_exact*: ⟦ *v* = *maxmin t*; *a* ≤ *v* ∧ *v* ≤ *b* ⟧ ⟹ *ab_max' a b t* = *v*
⟨*proof*⟩

Searched trees

Hard:

fun *abt_max* :: (*'a*::*linorder*) ⇒ *'a* ⇒ *'a tree* ⇒ *'a tree* **and** *abt_maxs abt_min abt_mins* **where**
abt_max a b (*Lf x*) = *Lf x* |
abt_max a b (*Nd ts*) = *Nd* (*abt_maxs a b ts*) |

abt_maxs a b [] = [] |
abt_maxs a b (*t#ts*) = (*let u* = *abt_min a b t*; *a'* = *max a* (*ab_min a b t*) *in*
u # (*if a'* ≥ *b then* [] *else abt_maxs a' b ts*)) |

abt_min a b (*Lf x*) = *Lf x* |

abt_min a b (*Nd ts*) = *Nd* (*abt_mins a b ts*) |

abt_mins a b [] = [] |
abt_mins a b (*t*#*ts*) = (*let u* = *abt_max a b t*; *b'* = *min b* (*ab_max a b t*) *in*
u # (*if b'* ≤ *a then* [] *else abt_mins a b' ts*))

Soft:

fun *abt_max'* :: (*'a::bounded_linorder*) ⇒ *'a* ⇒ *'a tree* ⇒ *'a tree* **and** *abt_maxs'*
abt_min' abt_mins' **where**
abt_max' a b (*Lf x*) = *Lf x* |
abt_max' a b (*Nd ts*) = *Nd* (*abt_maxs' a b* ⊥ *ts*) |

abt_maxs' a b m [] = [] |
abt_maxs' a b m (*t*#*ts*) =
(*let u* = *abt_min'* (*max m a*) *b t*; *m'* = *max m* (*ab_min'* (*max m a*) *b t*) *in*
u # (*if m'* ≥ *b then* [] *else abt_maxs' a b m' ts*)) |

abt_min' a b (*Lf x*) = *Lf x* |
abt_min' a b (*Nd ts*) = *Nd* (*abt_mins' a b* ⊤ *ts*) |

abt_mins' a b m [] = [] |
abt_mins' a b m (*t*#*ts*) =
(*let u* = *abt_max' a* (*min m b*) *t*; *m'* = *min m* (*ab_max' a* (*min m b*) *t*) *in*
u # (*if m'* ≤ *a then* [] *else abt_mins' a b m' ts*))

lemma *abt_max'_abt_max*:
shows *a* < *b* ⟹ *abt_max' a b t* = *abt_max a b t*
and *max m a* < *b* ⟹ *abt_maxs' a b m ts* = *abt_maxs* (*max m a*) *b ts*
and *a* < *b* ⟹ *abt_min' a b t* = *abt_min a b t*
and *a* < *min m b* ⟹ *abt_mins' a b m ts* = *abt_mins a* (*min m b*) *ts*
⟨*proof*⟩

An annotated tree of *ab* calls with the *a*,*b* window.

datatype *'a tri* = *Ma 'a 'a 'a tr* | *Mi 'a 'a 'a tr*
and *'a tr* = *No 'a tri list* | *Le 'a*

fun *abtr_max* :: (*'a::linorder*) ⇒ *'a* ⇒ *'a tree* ⇒ *'a tri* **and** *abtr_maxs abtr_min*
abtr_mins **where**
abtr_max a b (*Lf x*) = *Ma a b* (*Le x*) |
abtr_max a b (*Nd ts*) = *Ma a b* (*No* (*abtr_maxs a b ts*)) |

abtr_maxs a b [] = [] |
abtr_maxs a b (*t*#*ts*) = (*let u* = *abtr_min a b t*; *a'* = *max a* (*ab_min a b t*) *in*
u # (*if a'* ≥ *b then* [] *else abtr_maxs a' b ts*)) |

abtr_min a b (*Lf x*) = *Mi a b* (*Le x*) |
abtr_min a b (*Nd ts*) = *Mi a b* (*No* (*abtr_mins a b ts*)) |

abtr_mins a b [] = [] |

$abtr\_mins\ a\ b\ (t\#ts) = (let\ u = abtr\_max\ a\ b\ t;\ b' = min\ b\ (ab\_max\ a\ b\ t)\ in$
$u\ \#\ (if\ b' \le a\ then\ []\ else\ abtr\_mins\ a\ b'\ ts))$

For better readability get rid of *ereal*:

fun $de :: ereal \Rightarrow real$ **where**
$de\ (ereal\ x) = x\ |$
$de\ PInfty = 100\ |$
$de\ MInfty = -100$

fun *detri* **and** *detr* **where**
$detri\ (Ma\ a\ b\ t) = Ma\ (de\ a)\ (de\ b)\ (detr\ t)\ |$
$detri\ (Mi\ a\ b\ t) = Mi\ (de\ a)\ (de\ b)\ (detr\ t)\ |$
$detr\ (No\ ts) = No\ (map\ detri\ ts)\ |$
$detr\ (Le\ x) = Le\ (de\ x)$

Example in Knuth and Moore. Evaluation confirms that all subtrees u are pruned.

value *let*
$t11 = Nd[Nd[Lf\ 3,Lf\ 1,Lf\ 4],\ Nd[Lf\ 1,t],\ Nd[Lf\ 2,Lf\ 6,Lf\ 5]];$
$t12 = Nd[Nd[Lf\ 3,Lf\ 5,Lf\ 8],\ u];\ t13 = Nd[Nd[Lf\ 8,Lf\ 4,Lf\ 6],\ u];$
$t21 = Nd[Nd[Lf\ 3,Lf\ 2],Nd[Lf\ 9,Lf\ 5,Lf\ 0],Nd[Lf\ 2,u]];$
$t31 = Nd[Nd[Lf\ 0,u],Nd[Lf\ 4,Lf\ 9,Lf\ 4],Nd[Lf\ 4,u]];$
$t32 = Nd[Nd[Lf\ 2,u],Nd[Lf\ 7,Lf\ 8,Lf\ 1],Nd[Lf\ 6,Lf\ 4,Lf\ 0]];$
$t = Nd[Nd[t11,\ t12,\ t13],\ Nd[t21,u],\ Nd[t31,t32,u]]$
$in\ (ab\_max\ (-\infty::ereal)\ \infty\ t, abt\_max\ (-\infty::ereal)\ \infty\ t, detri(abtr\_max\ (-\infty::ereal)$
$\infty\ t))$

Soft, generalized, attempts

Attempts to prove correct General version due to Junkang Li et al.

This first version (not worth following!) stops the list iteration as soon as $max\ m\ a \ge b$ (I call this "delayed test", it complicates proofs a little.) and the initial value is fixed a (not $i0/1$)

fun $abil0' :: ('a::bounded\_linorder)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abils0'\ abil1'\ abils1'$
where
$abil0'\ (Lf\ x)\ a\ b = x\ |$
$abil0'\ (Nd\ ts)\ a\ b = abils0'\ ts\ a\ b\ a\ |$

$abils0'\ []\ a\ b\ m = m\ |$
$abils0'\ (t\#ts)\ a\ b\ m =$
$(if\ max\ m\ a \ge b\ then\ m\ else\ abils0'\ ts\ (max\ m\ a)\ b\ (max\ m\ (abil1'\ t\ b\ (max\ m$
$a))))\ |$

$abil1'\ (Lf\ x)\ a\ b = x\ |$
$abil1'\ (Nd\ ts)\ a\ b = abils1'\ ts\ a\ b\ a\ |$

$abils1'\ []\ a\ b\ m = m\ |$
$abils1'\ (t\#ts)\ a\ b\ m =$

(if min m a $\le$ b $then$ m $else$ $abils1'$ ts $(min$ m $a)$ b $(min$ m $(abil0'$ t b $(min$ m $a))))$

lemma $abils0'\_ge\_i$: $abils0'$ ts a b $i \ge i$
⟨*proof*⟩

lemma $abils1'\_le\_i$: $abils1'$ ts a b $i \le i$
⟨*proof*⟩

lemma $fishburn\_abil01'$:
shows $a < b \Longrightarrow fishburn$ a b $(maxmin$ $t)$ $(abil0'$ t a $b)$
and $a < b \Longrightarrow i < b \Longrightarrow fishburn$ $(max$ a $i)$ b $(maxmin$ $(Nd$ $ts))$ $(abils0'$ ts a b $i)$
and $a > b \Longrightarrow fishburn$ b a $(minmax$ $t)$ $(abil1'$ t a $b)$
and $a > b \Longrightarrow i > b \Longrightarrow fishburn$ b $(min$ a $i)$ $(minmax$ $(Nd$ $ts))$ $(abils1'$ ts a b $i)$
⟨*proof*⟩

This second computes the value of t before deciding if it needs to look at ts as well. This simplifies the proof (also in other versions, independently of initialization). The initial value is not fixed but determined by $i0/1$. The "real" constraint on $i0/1$ is commented out and replaced by the simplified value a.

locale $LeftSoft$ =
fixes $i0$ $i1$ :: $'a::bounded\_linorder$ $tree$ $list \Rightarrow 'a \Rightarrow 'a$
assumes $i0$: $i0$ ts $a \le a$ — max a (maxmin (Nd ts)) **and** $i1$: $i1$ ts $a \ge a$— min a (minmax (Nd ts))
begin

fun $abil0'$ :: $('a::bounded\_linorder)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abils0'$ $abil1'$ $abils1'$
where
$abil0'$ $(Lf$ $x)$ a $b = x$ |
$abil0'$ $(Nd$ $ts)$ a $b = abils0'$ ts a b $(i0$ ts $a)$ |

$abils0'$ [] a b $m = m$ |
$abils0'$ $(t\#ts)$ a b m =
$(let$ $m' = max$ m $(abil1'$ t b $(max$ m $a))$ in if $m' \ge b$ $then$ m' $else$ $abils0'$ ts a b $m')$ |

$abil1'$ $(Lf$ $x)$ a $b = x$ |
$abil1'$ $(Nd$ $ts)$ a $b = abils1'$ ts a b $(i1$ ts $a)$ |

$abils1'$ [] a b $m = m$ |
$abils1'$ $(t\#ts)$ a b m =
$(let$ $m' = min$ m $(abil0'$ t b $(min$ m $a))$ in if $m' \le b$ $then$ m' $else$ $abils1'$ ts a b $m')$

lemma $abils0'\_ge\_i$: $abils0'$ ts a b $i \ge i$
⟨*proof*⟩

lemma $abils1'\_le\_i$: $abils1'\ ts\ a\ b\ i \leq i$
⟨*proof*⟩

Generalizations that don't seem to work: a) $max\ a\ i \rightarrow max\ (max\ a\ (maxmin\ (Nd\ ts)))\ i$ b) ?

lemma $fishburn\_abil01'$:
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t) \qquad (abil0'\ t\ a\ b)$
and $a < b \Longrightarrow i < b \Longrightarrow fishburn\ (max\ a\ i)\ b\ (maxmin\ (Nd\ ts))\ (abils0'\ ts\ a\ b\ i)$
and $a > b \Longrightarrow fishburn\ b\ a\ (minmax\ t) \qquad (abil1'\ t\ a\ b)$
and $a > b \Longrightarrow i > b \Longrightarrow fishburn\ b\ (min\ a\ i)\ (minmax\ (Nd\ ts))\ (abils1'\ ts\ a\ b\ i)$
⟨*proof*⟩

Note the $a \leq b$ instead of the $a < b$ in theorem $fishburn\_abir01'$:

lemma $abil0'lb\_ub$:
shows $a \leq b \Longrightarrow lb\_ub\ a\ b\ (maxmin\ t)\ (abil0'\ t\ a\ b)$
and $a \leq b \Longrightarrow lb\_ub\ a\ b\ (max\ i\ (maxmin\ (Nd\ ts)))\ (abils0'\ ts\ a\ b\ i)$
and $a \geq b \Longrightarrow lb\_ub\ b\ a\ (minmax\ t)\ (abil1'\ t\ a\ b)$
and $a \geq b \Longrightarrow lb\_ub\ b\ a\ (min\ i\ (minmax\ (Nd\ ts)))\ (abils1'\ ts\ a\ b\ i)$
⟨*proof*⟩

lemma $abil0'\_exact\_less$: $[\![\ a<b;\ v = maxmin\ t;\ a \leq v \wedge v \leq b\ ]\!] \Longrightarrow abil0'\ t\ a\ b = v$
⟨*proof*⟩

lemma $abil0'\_exact$: $[\![\ v = maxmin\ t;\ a \leq v \wedge v \leq b]\!] \Longrightarrow abil0'\ t\ a\ b = v$
⟨*proof*⟩

end

Transposition Table / Graph / Repeated AB

lemma ab_twice_lb:
$[\![\ \forall\ a\ b.\ fishburn\ a\ b\ (maxmin\ t)\ (abf\ a\ b\ t);\ b \leq abf\ a\ b\ t;\ max\ a'\ (abf\ a\ b\ t) < b'\ ]\!] \Longrightarrow$
$fishburn\ a'\ b'\ (maxmin\ t)\ (abf\ (max\ a'\ (abf\ a\ b\ t))\ b'\ t)$
⟨*proof*⟩

lemma ab_twice_ub:
$[\![\ \forall\ a\ b.\ fishburn\ a\ b\ (maxmin\ t)\ (abf\ a\ b\ t);\ abf\ a\ b\ t \leq a;\ min\ b'\ (abf\ a\ b\ t) > a'\ ]\!] \Longrightarrow$
$fishburn\ a'\ b'\ (maxmin\ t)\ (abf\ a'\ (min\ b'\ (abf\ a\ b\ t))\ t)$
⟨*proof*⟩

But what does a narrower window achieve? Less precise bounds but prefix of search space. For fail-hard and fail-soft.

fun *prefix prefixs* **where**

prefix (*Lf x*) (*Lf y*) = (*x*=*y*) |
prefix (*Nd ts*) (*Nd us*) = *prefixs ts us* |
prefix _ _ = *False* |

prefixs [] *us* = *True* |
prefixs (*t*#*ts*) (*u*#*us*) = (*prefix t u* ∧ *prefixs ts us*) |
prefixs _ _ = *False*

lemma *fishburn_ab_max_windows*:
shows ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_max a' b' t*) (*ab_max a b t*)
and ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_maxs a' b' ts*) (*ab_maxs a b ts*)
and ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_min a' b' t*) (*ab_min a b t*)
and ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_mins a' b' ts*) (*ab_mins a b ts*)
⟨*proof*⟩

lemma *abt_max_prefix_windows*:
shows ⟦ $a' \le a$; $b \le b'$ ⟧ ⟹ *prefix* (*abt_max a b t*) (*abt_max a' b' t*)
and ⟦ $a' \le a$; $b \le b'$ ⟧ ⟹ *prefixs* (*abt_maxs a b ts*) (*abt_maxs a' b' ts*)
and ⟦ $a' \le a$; $b \le b'$ ⟧ ⟹ *prefix* (*abt_min a b t*) (*abt_min a' b' t*)
and ⟦ $a' \le a$; $b \le b'$ ⟧ ⟹ *prefixs* (*abt_mins a b ts*) (*abt_mins a' b' ts*)
⟨*proof*⟩

lemma *fishburn_ab_max'_windows*:
shows ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_max' a' b' t*) (*ab_max' a b t*)
and ⟦ *max m a* $< b$; $a' \le a$; $b \le b'$; $m' \le m$ ⟧ ⟹ *fishburn* (*max m a*) *b* (*ab_maxs' a' b' m' ts*) (*ab_maxs' a b m ts*)
and ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *fishburn a b* (*ab_min' a' b' t*) (*ab_min' a b t*)
and ⟦ $a <$ *min m b*; $a' \le a$; $b \le b'$; $m \le m'$ ⟧ ⟹ *fishburn a* (*min m b*) (*ab_mins' a' b' m' ts*) (*ab_mins' a b m ts*)
⟨*proof*⟩

Example of reduced search space:

lemma *let* $a = 0$; $b = (1::$*ereal*$)$; $a' = 0$; $b' = 2$; $t =$ *Nd* [*Lf* 1, *Lf* 0]
in abt_max' a b t = *Nd* [*Lf* 1] ∧ *abt_max' a' b' t* = *t*
⟨*proof*⟩

lemma *abt_max'_prefix_windows*:
shows ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *prefix* (*abt_max' a b t*) (*abt_max' a' b' t*)
and ⟦ *max m a* $< b$; $a' \le a$; $b \le b'$; $m' \le m$ ⟧ ⟹ *prefixs* (*abt_maxs' a b m ts*) (*abt_maxs' a' b' m' ts*)
and ⟦ $a < b$; $a' \le a$; $b \le b'$ ⟧ ⟹ *prefix* (*abt_min' a b t*) (*abt_min' a' b' t*)
and ⟦ $a <$ *min m b*; $a' \le a$; $b \le b'$; $m \le m'$ ⟧ ⟹ *prefixs* (*abt_mins' a b m ts*) (*abt_mins' a' b' m' ts*)
⟨*proof*⟩

2.4.2 From the Right

The literature uniformly considers iteration from the left only. Iteration from the right is technically simpler but needs to go through all successors, which means generating all of them. This is typically done anyway to reorder them based on heuristic evaluations. This rules out an infinite list of successors, but it is unclear if there are any applications.

Naming convention: 0 = max, 1 = min

Hard

fun *abr0* :: (*'a*::*linorder*)*tree* ⇒ *'a* ⇒ *'a* ⇒ *'a* **and** *abrs0* **and** *abr1* **and** *abrs1*
where
abr0 (*Lf x*) *a b* = *x* |
abr0 (*Nd ts*) *a b* = *abrs0 ts a b* |

abrs0 [] *a b* = *a* |
abrs0 (*t#ts*) *a b* = (*let m* = *abrs0 ts a b in if m* ≥ *b then m else max* (*abr1 t b m*) *m*) |

abr1 (*Lf x*) *a b* = *x* |
abr1 (*Nd ts*) *a b* = *abrs1 ts a b* |

abrs1 [] *a b* = *a* |
abrs1 (*t#ts*) *a b* = (*let m* = *abrs1 ts a b in if m* ≤ *b then m else min* (*abr0 t b m*) *m*)

lemma *abrs0_ge_a*: *abrs0 ts a b* ≥ *a*
⟨*proof*⟩

lemma *abrs1_le_a*: *abrs1 ts a b* ≤ *a*
⟨*proof*⟩

theorem *abr01_mm*:
shows *mm a* (*abr0 t a b*) *b* = *mm a* (*maxmin t*) *b*
and *mm a* (*abrs0 ts a b*) *b* = *mm a* (*maxmin* (*Nd ts*)) *b*
and *mm b* (*abr1 t a b*) *a* = *mm b* (*minmax t*) *a*
and *mm b* (*abrs1 ts a b*) *a* = *mm b* (*minmax* (*Nd ts*)) *a*
⟨*proof*⟩

As a corollary:

corollary *knuth_abr01_cor*: *a* < *b* ⟹ *knuth a b* (*maxmin t*) (*abr0 t a b*)
⟨*proof*⟩

corollary *maxmin_mm_abr0*: ⟦ *a* ≤ *maxmin t*; *maxmin t* ≤ *b* ⟧ ⟹ *maxmin t* = *mm a* (*abr0 t a b*) *b*
⟨*proof*⟩
corollary *maxmin_mm_abrs0*: ⟦ *a* ≤ *maxmin* (*Nd ts*); *maxmin* (*Nd ts*) ≤ *b* ⟧

$\Longrightarrow$ *maxmin* (*Nd ts*) = *mm a* (*abrs*0 *ts a b*) *b*
⟨*proof*⟩

The stronger *fishburn* spec:

Needs $a<b$.

theorem *fishburn_abr*01:
shows $a < b \Longrightarrow$ *fishburn a b* (*maxmin t*) (*abr*0 *t a b*)
and $a < b \Longrightarrow$ *fishburn a b* (*maxmin* (*Nd ts*)) (*abrs*0 *ts a b*)
and $a > b \Longrightarrow$ *fishburn b a* (*minmax t*) (*abr*1 *t a b*)
and $a > b \Longrightarrow$ *fishburn b a* (*minmax* (*Nd ts*)) (*abrs*1 *ts a b*)
⟨*proof*⟩

Above lemma does not work for $a = b$ and $a > b$. Not fishburn: *abr*0 $\leq$ *a* but not *maxmin* $\leq$ *abr*0. Not knuth: *abr*0 $\leq$ *a* but not *maxmin* $\leq$ *a*

lemma *let a* = 0::*ereal*; *t* = *Nd* [*Lf* 1, *Lf* 0] *in abr*0 *t a a* = 0 ∧ *maxmin t* = 1
⟨*proof*⟩
lemma *let a* = 0::*ereal*; *b* = −1; *t* = *Nd* [*Lf* 1, *Lf* 0] *in abr*0 *t a b* = 0 ∧ *maxmin t* = 1
⟨*proof*⟩

The following lemma does not follow from *fishburn* because of the weaker assumption $a \leq b$ that is required for the later exactness lemma.

lemma *abr0_le_b*: ⟦ $a \leq b$; *maxmin t* $\leq b$ ⟧ $\Longrightarrow$ *abr*0 *t a b* $\leq b$
and ⟦ $a \leq b$; *maxmin* (*Nd ts*) $\leq b$ ⟧ $\Longrightarrow$ *abrs*0 *ts a b* $\leq b$
and ⟦ $b \leq$ *minmax t*; $b \leq a$ ⟧ $\Longrightarrow$ $b \leq$ *abr*1 *t a b*
and ⟦ $b \leq$ *minmax* (*Nd ts*); $b \leq a$ ⟧ $\Longrightarrow$ $b \leq$ *abrs*1 *ts a b*
⟨*proof*⟩

lemma *abr0_exact_less*:
assumes $a < b$ v = *maxmin t* $a \leq v \wedge v \leq b$
shows *abr*0 *t a b* = *v*
⟨*proof*⟩

lemma *abr0_exact*:
assumes v = *maxmin t* $a \leq v \wedge v \leq b$
shows *abr*0 *t a b* = *v*
⟨*proof*⟩

Another proof:

lemma *abr0_exact2*:
assumes v = *maxmin t* $a \leq v \wedge v \leq b$
shows *abr*0 *t a b* = *v*
⟨*proof*⟩

Soft

Starting at ⊥ (after Fishburn)

fun *abr*0′ :: (′*a*::*bounded_linorder*)*tree* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* **and** *abrs*0′ **and** *abr*1′ **and** *abrs*1′ **where**

$abr0'$ $(Lf\ x)$ a $b = x$ |
$abr0'$ $(Nd\ ts)$ a $b = abrs0'$ ts a b |

$abrs0'$ [] a $b = \bot$ |
$abrs0'$ $(t\#ts)$ a $b = (let\ m = abrs0'\ ts\ a\ b\ in\ if\ m \geq b\ then\ m\ else\ max\ (abr1'\ t\ b$ $(max\ m\ a))\ m)$ |

$abr1'$ $(Lf\ x)$ a $b = x$ |
$abr1'$ $(Nd\ ts)$ a $b = abrs1'$ ts a b |

$abrs1'$ [] a $b = \top$ |
$abrs1'$ $(t\#ts)$ a $b = (let\ m = abrs1'\ ts\ a\ b\ in\ if\ m \leq b\ then\ m\ else\ min\ (abr0'\ t\ b$ $(min\ m\ a))\ m)$

theorem *fishburn_abr01'*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t)\quad (abr0'\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ (Nd\ ts))\ (abrs0'\ ts\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (minmax\ t)\quad (abr1'\ t\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (minmax\ (Nd\ ts))\ (abrs1'\ ts\ a\ b)$
⟨*proof*⟩

Same as for $abr0$: Above lemma does not work for $a = b$ and $a > b$. Not fishburn: $abr0' \leq a$ but not $maxmin \leq abr0'$. Not knuth: $abr0' \leq a$ but not $maxmin \leq a$

lemma $let\ a = 0{::}ereal;\ t = Nd\ [Lf\ 1,\ Lf\ 0]\ in\ abr0'\ t\ a\ a = 0 \wedge maxmin\ t = 1$
⟨*proof*⟩
lemma $let\ a = 0{::}ereal;\ b = -1;\ t = Nd\ [Lf\ 1,\ Lf\ 0]\ in\ abr0'\ t\ a\ b = 0 \wedge maxmin$ $t = 1$
⟨*proof*⟩

Fails for a=b=-1 and t = Nd []

theorem *fishburn2_abr01_abr01'*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (abr0'\ t\ a\ b)\quad (abr0\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (abrs0'\ ts\ a\ b)\ (abrs0\ ts\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (abr1'\ t\ a\ b)\quad (abr1\ t\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (abrs1'\ ts\ a\ b)\ (abrs1\ ts\ a\ b)$
⟨*proof*⟩

Towards 'exactness':

No need for restricting a,b, but only corollaries:

corollary *abr0'_mm*: $mm\ a\ (abr0'\ t\ a\ b)\ b = mm\ a\ (maxmin\ t)\ b$
⟨*proof*⟩
corollary *abrs0'_mm*: $mm\ a\ (abrs0'\ ts\ a\ b)\ b = mm\ a\ (maxmin\ (Nd\ ts))\ b$
⟨*proof*⟩
corollary *abr1'_mm*: $mm\ b\ (abr1'\ t\ a\ b)\ a = mm\ b\ (minmax\ t)\ a$
⟨*proof*⟩
corollary *abrs1'_mm*: $mm\ b\ (abrs1'\ ts\ a\ b)\ a = mm\ b\ (minmax\ (Nd\ ts))\ a$
⟨*proof*⟩

corollary *li*1: $[\![a \le maxmin\ t;\ \ maxmin\ t \le b]\!] \Longrightarrow mm\ a\ (abr0'\ t\ a\ b)\ b = maxmin\ t$
⟨*proof*⟩

Note the $a \le b$ instead of the $a < b$ in $a < b \Longrightarrow abr0'\ t\ a\ b \le maxmin\ t$ (mod a,b)
$a < b \Longrightarrow abrs0'\ ts\ a\ b \le maxmin\ (Nd\ ts)$ (mod a,b)
$b < a \Longrightarrow abr1'\ t\ a\ b \le minmax\ t$ (mod b,a)
$b < a \Longrightarrow abrs1'\ ts\ a\ b \le minmax\ (Nd\ ts)$ (mod b,a):

lemma *abr*01′*lb_ub*:
shows $a \le b \Longrightarrow lb\_ub\ a\ b\ (maxmin\ t)\quad (abr0'\ t\ a\ b)$
and $a \le b \Longrightarrow lb\_ub\ a\ b\ (maxmin\ (Nd\ ts))\ (abrs0'\ ts\ a\ b)$
and $a \ge b \Longrightarrow lb\_ub\ b\ a\ (minmax\ t)\quad (abr1'\ t\ a\ b)$
and $a \ge b \Longrightarrow lb\_ub\ b\ a\ (minmax\ (Nd\ ts))\ (abrs1'\ ts\ a\ b)$
⟨*proof*⟩

lemma *abr*0′*_exact_less*: $[\![\ a{<}b;\ v = maxmin\ t;\ a \le v \wedge v \le b\ ]\!] \Longrightarrow abr0'\ t\ a\ b = v$
⟨*proof*⟩

lemma *abr*0′*_exact*: $[\![\ v = maxmin\ t;\ a \le v \wedge v \le b]\!] \Longrightarrow abr0'\ t\ a\ b = v$
⟨*proof*⟩

Also returning the searched tree

Hard:

fun *abtr*0 :: (′*a*::*linorder*) *tree* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ∗ ′*a tree* **and** *abtrs*0 **and** *abtr*1 **and** *abtrs*1 **where**
*abtr*0 (*Lf x*) *a b* = (*x*, *Lf x*) |
*abtr*0 (*Nd ts*) *a b* = (*let* (*m*,*us*) = *abtrs*0 *ts a b in* (*m*, *Nd us*)) |

*abtrs*0 [] *a b* = (*a*,[]) |
*abtrs*0 (*t*#*ts*) *a b* = (*let* (*m*,*us*) = *abtrs*0 *ts a b in*
if m ≥ *b then* (*m*,*us*) *else let* (*n*,*u*) = *abtr*1 *t b m in* (*max n m*,*u*#*us*)) |

*abtr*1 (*Lf x*) *a b* = (*x*, *Lf x*) |
*abtr*1 (*Nd ts*) *a b* = (*let* (*m*,*us*) = *abtrs*1 *ts a b in* (*m*, *Nd us*)) |

*abtrs*1 [] *a b* = (*a*,[]) |
*abtrs*1 (*t*#*ts*) *a b* = (*let* (*m*,*us*) = *abtrs*1 *ts a b in*
if m ≤ *b then* (*m*,*us*) *else let* (*n*,*u*) = *abtr*0 *t b m in* (*min n m*,*u*#*us*))

Soft:

fun *abtr*0′ :: (′*a*::*bounded_linorder*) *tree* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ∗ ′*a tree* **and** *abtrs*0′ **and** *abtr*1′ **and** *abtrs*1′ **where**
*abtr*0′ (*Lf x*) *a b* = (*x*, *Lf x*) |
*abtr*0′ (*Nd ts*) *a b* = (*let* (*m*,*us*) = *abtrs*0′ *ts a b in* (*m*, *Nd us*)) |

$abtrs0'$ [] a b = $(\bot,[])$ |
$abtrs0'$ $(t\#ts)$ a b = $(let\ (m,us) = abtrs0'\ ts\ a\ b\ in$
  $if\ m \geq b\ then\ (m,us)\ else\ let\ (n,u) = abtr1'\ t\ b\ (max\ m\ a)\ in\ (max\ n\ m,u\#us))$ |

$abtr1'$ $(Lf\ x)$ a b = $(x,\ Lf\ x)$ |
$abtr1'$ $(Nd\ ts)$ a b = $(let\ (m,us) = abtrs1'\ ts\ a\ b\ in\ (m,\ Nd\ us))$ |

$abtrs1'$ [] a b = $(\top,[])$ |
$abtrs1'$ $(t\#ts)$ a b = $(let\ (m,us) = abtrs1'\ ts\ a\ b\ in$
  $if\ m \leq b\ then\ (m,us)\ else\ let\ (n,u) = abtr0'\ t\ b\ (min\ m\ a)\ in\ (min\ n\ m,u\#us))$

lemma *fst_abtr01*:
shows $fst(abtr0\ t\ a\ b) = abr0\ t\ a\ b$
and $fst(abtrs0\ ts\ a\ b) = abrs0\ ts\ a\ b$
and $fst(abtr1\ t\ a\ b) = abr1\ t\ a\ b$
and $fst(abtrs1\ ts\ a\ b) = abrs1\ ts\ a\ b$
⟨*proof*⟩

lemma *fst_abtr01'*:
shows $fst(abtr0'\ t\ a\ b) = abr0'\ t\ a\ b$
and $fst(abtrs0'\ ts\ a\ b) = abrs0'\ ts\ a\ b$
and $fst(abtr1'\ t\ a\ b) = abr1'\ t\ a\ b$
and $fst(abtrs1'\ ts\ a\ b) = abrs1'\ ts\ a\ b$
⟨*proof*⟩

lemma *snd_abtr01'_abtr01*:
shows $a < b \Longrightarrow snd(abtr0'\ t\ a\ b) = snd(abtr0\ t\ a\ b)$
and $a < b \Longrightarrow snd(abtrs0'\ ts\ a\ b) = snd(abtrs0\ ts\ a\ b)$
and $a > b \Longrightarrow snd(abtr1'\ t\ a\ b) = snd(abtr1\ t\ a\ b)$
and $a > b \Longrightarrow snd(abtrs1'\ ts\ a\ b) = snd(abtrs1\ ts\ a\ b)$
⟨*proof*⟩

Generalized

General version due to Junkang Li et al.:

locale *SoftGeneral* =
fixes $i0$ $i1$ :: $'a::bounded\_linorder\ tree\ list \Rightarrow 'a \Rightarrow 'a$
assumes $i0$: $i0\ ts\ a \leq max\ a\ (maxmin(Nd\ ts))$ **and** $i1$: $i1\ ts\ a \geq min\ a\ (minmax\ (Nd\ ts))$
begin

fun $abir0'$ :: $('a::bounded\_linorder)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abirs0'$ **and** $abir1'$ **and** $abirs1'$ **where**
$abir0'$ $(Lf\ x)$ a b = x |
$abir0'$ $(Nd\ ts)$ a b = $abirs0'\ (i0\ ts\ a)\ ts\ a\ b$ |

$abirs0'$ i [] a b = i |
$abirs0'$ i $(t\#ts)$ a b =

$(let\ m = abirs0'\ i\ ts\ a\ b\ in\ if\ m \geq b\ then\ m\ else\ max\ (abir1'\ t\ b\ (max\ m\ a))\ m)$ |

$abir1'\ (Lf\ x)\ a\ b = x$ |
$abir1'\ (Nd\ ts)\ a\ b = abirs1'\ (i1\ ts\ a)\ ts\ a\ b$ |

$abirs1'\ i\ []\ a\ b = i$ |
$abirs1'\ i\ (t\#ts)\ a\ b =$
$(let\ m = abirs1'\ i\ ts\ a\ b\ in\ if\ m \leq b\ then\ m\ else\ min\ (abir0'\ t\ b\ (min\ m\ a))\ m)$

Unused:

lemma $abirs0'\_ge\_i$: $abirs0'\ i\ ts\ a\ b \geq i$
⟨*proof*⟩

lemma $abirs1'\_le\_i$: $abirs1'\ i\ ts\ a\ b \leq i$
⟨*proof*⟩

lemma $fishburn\_abir01'$:
shows $a < b \Longrightarrow fishburn\ a\ b\ (maxmin\ t)\quad (abir0'\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (max\ i\ (maxmin\ (Nd\ ts)))\ (abirs0'\ i\ ts\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (minmax\ t)\quad (abir1'\ t\ a\ b)$
and $a > b \Longrightarrow fishburn\ b\ a\ (min\ i\ (minmax\ (Nd\ ts)))\ (abirs1'\ i\ ts\ a\ b)$
⟨*proof*⟩

Note the $a \leq b$ instead of the $a < b$ in $a < b \Longrightarrow abir0'\ t\ a\ b \leq maxmin\ t\ (\mathrm{mod}\ a,b)$
$a < b \Longrightarrow abirs0'\ i\ ts\ a\ b \leq max\ i\ (maxmin\ (Nd\ ts))\ (\mathrm{mod}\ a,b)$
$b < a \Longrightarrow abir1'\ t\ a\ b \leq minmax\ t\ (\mathrm{mod}\ b,a)$
$b < a \Longrightarrow abirs1'\ i\ ts\ a\ b \leq min\ i\ (minmax\ (Nd\ ts))\ (\mathrm{mod}\ b,a)$:

lemma $abir0'lb\_ub$:
shows $a \leq b \Longrightarrow lb\_ub\ a\ b\ (maxmin\ t)\ (abir0'\ t\ a\ b)$
and $a \leq b \Longrightarrow lb\_ub\ a\ b\ (max\ i\ (maxmin\ (Nd\ ts)))\ (abirs0'\ i\ ts\ a\ b)$
and $a \geq b \Longrightarrow lb\_ub\ b\ a\ (minmax\ t)\ (abir1'\ t\ a\ b)$
and $a \geq b \Longrightarrow lb\_ub\ b\ a\ (min\ i\ (minmax\ (Nd\ ts)))\ (abirs1'\ i\ ts\ a\ b)$
⟨*proof*⟩

lemma $abir0'\_exact\_less$: $[\![\ a<b;\ v = maxmin\ t;\ a \leq v \wedge v \leq b\ ]\!] \Longrightarrow abir0'\ t\ a\ b = v$
⟨*proof*⟩

lemma $abir0'\_exact$: $[\![\ v = maxmin\ t;\ a \leq v \wedge v \leq b]\!] \Longrightarrow abir0'\ t\ a\ b = v$
⟨*proof*⟩

end

Now with explicit parameters $i0$ and $i1$ such that we can vary them:

fun $abir0' :: \_ \Rightarrow \_ \Rightarrow ('a::bounded\_linorder)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abirs0'$ **and** $abir1'$ **and** $abirs1'$ **where**
$abir0'\ i0\ i1\ (Lf\ x)\ a\ b = x$ |
$abir0'\ i0\ i1\ (Nd\ ts)\ a\ b = abirs0'\ i0\ i1\ (i0\ ts\ a)\ ts\ a\ b$ |

$abirs0'$ $i0$ $i1$ i [] a b = i |
$abirs0'$ $i0$ $i1$ i ($t\#ts$) a b =
(let m = $abirs0'$ $i0$ $i1$ i ts a b in if $m \geq b$ $then$ m $else$ max ($abir1'$ $i0$ $i1$ t b (max m a)) m) |

$abir1'$ $i0$ $i1$ (Lf x) a b = x |
$abir1'$ $i0$ $i1$ (Nd ts) a b = $abirs1'$ $i0$ $i1$ ($i1$ ts a) ts a b |

$abirs1'$ $i0$ $i1$ i [] a b = i |
$abirs1'$ $i0$ $i1$ i ($t\#ts$) a b =
(let m = $abirs1'$ $i0$ $i1$ i ts a b in if $m \leq b$ $then$ m $else$ min ($abir0'$ $i0$ $i1$ t b (min m a)) m)

First, the same theorem as in the locale *SoftGeneral*:

definition *bnd i0 i1* ≡
∀ *ts a*. *i0 ts a* ≤ *max a* (*maxmin*(*Nd ts*)) ∧ *i1 ts a* ≥ *min a* (*minmax* (*Nd ts*))

declare [[*unify_search_bound*=400,*unify_trace_bound*=400]]

lemma *fishburn_abir01'*:
shows $a < b \Longrightarrow$ *bnd i0 i1* $\Longrightarrow$ *fishburn a b* (*maxmin t*) (*abir0' i0 i1 t a b*)
and $a < b \Longrightarrow$ *bnd i0 i1* $\Longrightarrow$ *fishburn a b* (*max i* (*maxmin* (*Nd ts*))) (*abirs0' i0 i1 i ts a b*)
and $a > b \Longrightarrow$ *bnd i0 i1* $\Longrightarrow$ *fishburn b a* (*minmax t*) (*abir1' i0 i1 t a b*)
and $a > b \Longrightarrow$ *bnd i0 i1* $\Longrightarrow$ *fishburn b a* (*min i* (*minmax* (*Nd ts*))) (*abirs1' i0 i1 i ts a b*)
⟨*proof*⟩

Unused:

lemma *abirs0'_ge_i*: *abirs0' i0 i1 i ts a b* ≥ *i*
⟨*proof*⟩

lemma *abirs0'_eq_i*: *i* ≥ *b* $\Longrightarrow$ *abirs0' i0 i1 i ts a b* = *i*
⟨*proof*⟩

lemma *abirs1'_le_i*: *abirs1' i0 i1 i ts a b* ≤ *i*
⟨*proof*⟩

Monotonicity wrt the init functions, below/above a:

definition *bnd_mono i0 i1 i0' i1'* =
(∀ *ts a*. *i0' ts a* ≤ *a* ∧ *i1' ts a* ≥ *a* ∧ *i0 ts a* ≤ *i0' ts a* ∧ *i1 ts a* ≥ *i1' ts a*)

lemma *fishburn_abir0'_mono*:
shows $a < b \Longrightarrow$ *bnd_mono i0 i1 i0' i1'* $\Longrightarrow$ *fishburn a b* (*abir0' i0 i1 t a b*) (*abir0' i0' i1' t a b*)
and $a < b \Longrightarrow$ *bnd_mono i0 i1 i0' i1'* $\Longrightarrow$ *i* = *i0* (*ts0* @ *ts*) *a* $\Longrightarrow$
fishburn a b (*abirs0' i0 i1 i ts a b*) (*abirs0' i0' i1'* (*i0'* (*ts0* @ *ts*) *a*) *ts a b*)

and $a > b \Longrightarrow bnd\_mono\ i0\ i1\ i0'\ i1' \Longrightarrow fishburn\ b\ a\ (abir1'\ i0\ i1\ t\ a\ b)\ (abir1'\ i0'\ i1'\ t\ a\ b)$
and $a > b \Longrightarrow bnd\_mono\ i0\ i1\ i0'\ i1' \Longrightarrow i = i1\ (ts0@ts)\ a \Longrightarrow$
$fishburn\ b\ a\ (abirs1'\ i0\ i1\ i\ ts\ a\ b)\ (abirs1'\ i0'\ i1'\ (i1'\ (ts0\ @\ ts)\ a)\ ts\ a\ b)$
$\langle proof \rangle$

The $i0$ bound of a cannot be increased to $max\ a\ (maxmin(Nd\ ts)$ (as the theorem $fishburn\_abir0'$ might suggest). Problem: if $b \le i0\ a\ ts < i0'\ a\ ts$ then it can happen that $b \le abirs0'\ i0\ i1\ t\ a\ b < abirs0'\ i0'\ i1'\ t\ a\ b$, which violates *fishburn*.

value $let\ \ a = -\ \infty;\ b = 0{::}ereal;\ t = Nd\ [Lf\ (1{::}ereal)]\ in$
$(abir0'\ (\lambda ts\ a.\ max\ a\ (maxmin(Nd\ ts)))\ i1'\ t\ a\ b,$
$abir0'\ (\lambda ts\ a.\ max\ a\ (maxmin(Nd\ ts)){-}1)\ i1\ t\ a\ b)$

lemma $let\ \ a = -\ \infty;\ b = 0{::}ereal;\ ts = [Lf\ (1{::}ereal)]\ in$
$abirs0'\ (\lambda ts\ a.\ max\ a\ (maxmin(Nd\ ts)){-}1)\ (\lambda\_\ a.\ a{+}1)\ (max\ a\ (maxmin(Nd\ ts)){-}1)\ ts\ a\ b = 0$
$\langle proof \rangle$

2.5 Alpha-Beta for De Morgan Orders

2.5.1 From the Left, Fail-Hard

Like Knuth.

fun $ab\_negmax :: 'a \Rightarrow 'a \Rightarrow ('a{::}de\_morgan\_order)tree \Rightarrow 'a$ **and** $ab\_negmaxs$ **where**
$ab\_negmax\ a\ b\ (Lf\ x) = x\ |$
$ab\_negmax\ a\ b\ (Nd\ ts) = ab\_negmaxs\ a\ b\ ts\ |$

$ab\_negmaxs\ a\ b\ [] = a\ |$
$ab\_negmaxs\ a\ b\ (t\#ts) = (let\ a' = max\ a\ (-\ ab\_negmax\ (-b)\ (-a)\ t)\ in\ if\ a' \ge b\ then\ a'\ else\ ab\_negmaxs\ a'\ b\ ts)$

Via *foldl*. Wasteful: *foldl* consumes whole list.

definition $ab\_negmaxf :: ('a{::}de\_morgan\_order) \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **where**
$ab\_negmaxf\ b = (\lambda a\ t.\ if\ a \ge b\ then\ a\ else\ max\ a\ (-\ ab\_negmax\ (-b)\ (-a)\ t))$

lemma *foldl_ab_negmaxf_idemp*:
$b \le a \Longrightarrow foldl\ (ab\_negmaxf\ b)\ a\ ts = a$
$\langle proof \rangle$

lemma *ab_negmaxs_foldl*:
$(a{::}'a{::}de\_morgan\_order) < b \Longrightarrow ab\_negmaxs\ a\ b\ ts = foldl\ (ab\_negmaxf\ b)\ a\ ts$
$\langle proof \rangle$

Also returning the searched tree.

fun $abtl :: 'a \Rightarrow 'a \Rightarrow ('a{::}de\_morgan\_order)tree \Rightarrow 'a * ('a{::}de\_morgan\_order)tree$
and *abtls* **where**

abtl a b (Lf x) = (x, Lf x) |
abtl a b (Nd ts) = (let (m,us) = abtls a b ts in (m, Nd us)) |

abtls a b [] = (a,[]) |
abtls a b (t#ts) = (let (a′,u) = abtl (−b) (−a) t; a′ = max a (−a′) in
if a′ ≥ b then (a′,[u]) else let (n,us) = abtls a′ b ts in (n,u#us))

lemma *fst_abtl*:
shows *fst(abtl a b t) = ab_negmax a b t*
and *fst(abtls a b ts) = ab_negmaxs a b ts*
⟨*proof*⟩

Correctness Proofs

First, a very direct proof.

lemma *ab_negmaxs_ge_a*: *ab_negmaxs a b ts ≥ a*
⟨*proof*⟩

lemma *fishburn_val_ab_neg*:
shows *a < b ⟹ fishburn a b (negmax t) (ab_negmax (a) b t)*
and *a < b ⟹ fishburn a b (negmax (Nd ts)) (ab_negmaxs (a) b ts)*
⟨*proof*⟩

Now an indirect one by reduction to the min/max alpha-beta. Direct proof is simpler!

Relate ordinary and negmax ab:

theorem *ab_max_negmax*:
shows *ab_max a b t = ab_negmax a b (negate False t)*
and *ab_maxs a b ts = ab_negmaxs a b (map (negate True) ts)*
and *ab_min a b t = − ab_negmax (−b) (−a) (negate True t)*
and *ab_mins a b ts = − ab_negmaxs (−b) (−a) (map (negate False) ts)*
⟨*proof*⟩

corollary *fishburn_negmax_ab_negmax*: *a < b ⟹ fishburn a b (negmax t) (ab_negmax a b t)*
⟨*proof*⟩

lemma *ab_negmax_ab_le*:
shows *ab_negmax a b t = ab_le (≤) a b (negate False t)*
and *ab_negmaxs a b ts = ab_les (≤) a b (map (negate True) ts)*
and *ab_negmax a b t = − ab_le (≥) (−a) (−b) (negate True t)*
and *ab_negmaxs a b ts = − ab_les (≥) (−a) (−b) (map (negate False) ts)*
⟨*proof*⟩

Pointless? Weaker than fishburn and direct proof rather than corollary as via *ab_max_negmax*

Weaker max-min property. Proof: Case False one eqn chain, but dualized IH:

theorem
shows *ab_negmax_negmax2*: $max\ a\ (min\ (ab\_negmax\ a\ b\ t)\ b) = max\ a\ (min\ (negmax\ t)\ b)$
and *ab_negmaxs_maxs_neg3*: $a < b \Longrightarrow min\ (ab\_negmaxs\ a\ b\ ts)\ b = max\ a\ (min\ (negmax\ (Nd\ ts))\ b)$
⟨*proof*⟩

corollary *ab_negmax_negmax_cor2*: $ab\_negmax\ \bot\ \top\ t = negmax\ t$
⟨*proof*⟩

2.5.2 From the Left, Fail-Soft

After Fishburn

fun $ab\_negmax' :: 'a \Rightarrow 'a \Rightarrow ('a::de\_morgan\_order)tree \Rightarrow 'a$ **and** $ab\_negmaxs'$
where
$ab\_negmax'\ a\ b\ (Lf\ x) = x\ |$
$ab\_negmax'\ a\ b\ (Nd\ ts) = (ab\_negmaxs'\ a\ b\ \bot\ ts)\ |$

$ab\_negmaxs'\ a\ b\ m\ [] = m\ |$
$ab\_negmaxs'\ a\ b\ m\ (t\#ts) = (let\ m' = max\ m\ (-\ ab\_negmax'\ (-b)\ (-\ max\ m\ a)\ t)\ in$
$\quad if\ m' \geq b\ then\ m'\ else\ ab\_negmaxs'\ a\ b\ m'\ ts)$

lemma *ab_negmaxs'_ge_a*: $ab\_negmaxs'\ a\ b\ m\ ts \geq m$
⟨*proof*⟩

theorem *fishburn_val_ab_neg'*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ (ab\_negmax'\ a\ b\ t)$
and $max\ a\ m < b \Longrightarrow fishburn\ (max\ a\ m)\ b\ (negmax\ (Nd\ ts))\ (ab\_negmaxs'\ a\ b\ m\ ts)$
⟨*proof*⟩

theorem *fishburn_ab'_ab_neg*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (ab\_negmax'\ a\ b\ t)\ (ab\_negmax\ a\ b\ t)$
and $max\ m\ a < b \Longrightarrow fishburn\ a\ b\ (ab\_negmaxs'\ a\ b\ m\ ts)\ (ab\_negmaxs\ (max\ m\ a)\ b\ ts)$
⟨*proof*⟩

Another proof of *fishburn_negmax_ab_negmax*, just by transitivity:

corollary $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ (ab\_negmax\ a\ b\ t)$
⟨*proof*⟩

Now fail-soft with traversed trees.

fun $abtl'$ :: $'a \Rightarrow 'a \Rightarrow ('a::de\_morgan\_order)tree \Rightarrow 'a * ('a::de\_morgan\_order)tree$
and $abtls'$ **where**
$abtl'\ a\ b\ (Lf\ x) = (x,\ Lf\ x)\ |$
$abtl'\ a\ b\ (Nd\ ts) = (let\ (m,us) = abtls'\ a\ b\ \bot\ ts\ in\ (m,\ Nd\ us))\ |$

$abtls'\ a\ b\ m\ []\ = (m,[])\ |$
$abtls'\ a\ b\ m\ (t\#ts) = (let\ (m',u) = abtl'\ (-b)\ (-\ max\ m\ a)\ t;\ m' = max\ m\ (-m')\ in$
$\quad if\ m' \geq b\ then\ (m',[u])\ else\ let\ (n,us) = abtls'\ a\ b\ m'\ ts\ in\ (n,u\#us))$

lemma fst_abtl':
shows $fst(abtl'\ a\ b\ t) = ab\_negmax'\ a\ b\ t$
and $fst(abtls'\ a\ b\ m\ ts) = ab\_negmaxs'\ a\ b\ m\ ts$
⟨*proof*⟩

Fail-hard and fail-soft search the same part of the tree:

lemma $snd\_abtl'\_abtl$:
shows $a < b \Longrightarrow abtl'\ a\ b\ t = (ab\_negmax'\ a\ b\ t,\ snd(abtl\ a\ b\ t))$
and $max\ m\ a < b \Longrightarrow abtls'\ a\ b\ m\ ts = (ab\_negmaxs'\ a\ b\ m\ ts,\ snd(abtls\ (max\ m\ a)\ b\ ts))$
⟨*proof*⟩

min/*max* in *Lf*

fun $ab\_negmax2$ :: $('a::de\_morgan\_order) \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **and** $ab\_negmaxs2$ **where**
$ab\_negmax2\ a\ b\ (Lf\ x) = max\ a\ (min\ x\ b)\ |$
$ab\_negmax2\ a\ b\ (Nd\ ts) = ab\_negmaxs2\ a\ b\ ts\ |$

$ab\_negmaxs2\ a\ b\ []\ = a\ |$
$ab\_negmaxs2\ a\ b\ (t\#ts) = (let\ a' = -\ ab\_negmax2\ (-b)\ (-a)\ t\ in\ if\ a' = b\ then\ a'\ else\ ab\_negmaxs2\ a'\ b\ ts)$

lemma $ab\_negmax2\_max\_min\_negmax$:
shows $a < b \Longrightarrow ab\_negmax2\ a\ b\ t = max\ a\ (min\ (negmax\ t)\ b)$
and $a < b \Longrightarrow ab\_negmaxs2\ a\ b\ ts = max\ a\ (min\ (negmax\ (Nd\ ts))\ b)$
⟨*proof*⟩

corollary $ab\_negmax2\_bot\_top$: $ab\_negmax2\ \bot\ \top\ t = negmax\ t$
⟨*proof*⟩

Delayed test

Now a variant that delays the test to the next call of *ab_negmaxs*. Like Bird and Hughes' version, except that *ab_negmax3* does not cut off the return value.

fun $ab\_negmax3$ :: $('a::de\_morgan\_order) \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **and** $ab\_negmaxs3$ **where**
$ab\_negmax3\ a\ b\ (Lf\ x) = x\ |$

$ab\_negmax3\ a\ b\ (Nd\ ts) = ab\_negmaxs3\ a\ b\ ts$ |

$ab\_negmaxs3\ a\ b\ [] = a$ |
$ab\_negmaxs3\ a\ b\ (t\#ts) = (if\ a \geq b\ then\ a\ else\ ab\_negmaxs3\ (max\ a\ (-\ ab\_negmax3\ (-b)\ (-a)\ t))\ b\ ts)$

lemma $ab\_negmax3\_ab\_negmax$:
shows $a{<}b \Longrightarrow ab\_negmax3\ a\ b\ t = ab\_negmax\ a\ b\ t$
and $a{<}b \Longrightarrow ab\_negmaxs3\ a\ b\ ts = ab\_negmaxs\ a\ b\ ts$
⟨*proof*⟩

corollary $ab\_negmax3\_bot\_top$: $ab\_negmax3\ \bot\ \top\ t = negmax\ t$
⟨*proof*⟩

lemma $ab\_negmaxs3\_foldl$:
$ab\_negmaxs3\ a\ b\ ts = foldl\ (\lambda a\ t.\ if\ a \geq b\ then\ a\ else\ max\ a\ (-\ ab\_negmax3\ (-b)\ (-a)\ t))\ a\ ts$
⟨*proof*⟩

2.5.3 From the Right, Fail-Hard

fun $abr :: ('a{::}de\_morgan\_order)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abrs$ **where**
$abr\ (Lf\ x)\ a\ b = x$ |
$abr\ (Nd\ ts)\ a\ b = abrs\ ts\ a\ b$ |

$abrs\ []\ a\ b = a$ |
$abrs\ (t\#ts)\ a\ b = (let\ m = abrs\ ts\ a\ b\ in\ if\ m \geq b\ then\ m\ else\ max\ (-\ abr\ t\ (-b)\ (-m))\ m)$

lemma $Lf\_eq\_negateD$: $Lf\ x = negate\ f\ t \Longrightarrow t = Lf(if\ f\ then\ -x\ else\ x)$
⟨*proof*⟩

lemma $Nd\_eq\_negateD$: $Nd\ ts' = negate\ f\ t \Longrightarrow \exists\, ts.\ t = Nd\ ts \wedge ts' = map\ (negate\ (\neg f))\ ts$
⟨*proof*⟩

lemma $abr01\_negate$:
shows $abr0\ (negate\ f\ t)\ a\ b = -\ abr1\ (negate\ (\neg f)\ t)\ (-a)\ (-b)$
and $abrs0\ (map\ (negate\ f)\ ts)\ a\ b = -\ abrs1\ (map\ (negate\ (\neg f))\ ts)\ (-a)\ (-b)$
and $abr1\ (negate\ f\ t)\ a\ b = -\ abr0\ (negate\ (\neg f)\ t)\ (-a)\ (-b)$
and $abrs1\ (map\ (negate\ f)\ ts)\ a\ b = -\ abrs0\ (map\ (negate\ (\neg f))\ ts)\ (-a)\ (-b)$
⟨*proof*⟩

lemma abr_abr0:
shows $abr\ t\ a\ b = abr0\ (negate\ False\ t)\ a\ b$
and $abrs\ ts\ a\ b = abrs0\ (map\ (negate\ True)\ ts)\ a\ b$
⟨*proof*⟩

Relationship to *foldr*

fun *foldr* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ list \Rightarrow 'b$ **where**
$foldr\ f\ v\ [] = v$ |
$foldr\ f\ v\ (x\#xs) = f\ x\ (foldr\ f\ v\ xs)$

definition $abrsf\ b = (\lambda t\ m.\ if\ m \geq b\ then\ m\ else\ max\ (-\ abr\ t\ (-b)\ (-m))\ m)$

lemma *abrs_foldr*: $abrs\ ts\ a\ b = foldr\ (abrsf\ b)\ a\ ts$
⟨*proof*⟩

A direct (rather than mutually) recursive def of *abr*

lemma *abr_Nd_foldr*:
$abr\ (Nd\ ts)\ a\ b = foldr\ (abrsf\ b)\ a\ ts$
⟨*proof*⟩

Direct correctness proof of *foldr* version is no simpler than proof via *abr*/*abrs*:

lemma *fishburn_abr_foldr*: $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ (abr\ t\ a\ b)$
⟨*proof*⟩

The long proofs that follows are duplicated from the *bounded_linorder* section.

fishburn Proofs

lemma *abrs_ge_a*: $abrs\ ts\ a\ b \geq a$
⟨*proof*⟩

Automatic correctness proof, also works for *knuth* instead of *fishburn*:

corollary *fishburn_abr_negmax*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ \ (abr\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ (Nd\ ts))\ (abrs\ ts\ a\ b)$
⟨*proof*⟩

corollary *knuth_abr_negmax*: $a < b \Longrightarrow knuth\ a\ b\ (negmax\ t)\ (abr\ t\ a\ b)$
⟨*proof*⟩

corollary *abr_cor*: $abr\ t\ \bot\ \top = negmax\ t$
⟨*proof*⟩

Detailed *fishburn*2 proof (85 lines):

theorem *fishburn2_abr*:
shows $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ \ (abr\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ (Nd\ ts))\ (abrs\ ts\ a\ b)$
⟨*proof*⟩

Detailed *fishburn* proof (100 lines):

theorem *fishburn_abr*:

shows $a < b \Longrightarrow$ *fishburn a b* (*negmax t*) (*abr t a b*)
and $a < b \Longrightarrow$ *fishburn a b* (*negmax* (*Nd ts*)) (*abrs ts a b*)
⟨*proof*⟩

Explicit equational *knuth* proofs via min/max

Not mm, only min and max. Only min in abrs. $a<b$ required: a=1, b=-1, t=[]

theorem shows *abr_negmax3*: *max a* (*min* (*abr t a b*) *b*) = *max a* (*min* (*negmax t*) *b*)
and $a < b \Longrightarrow$ *min* (*abrs ts a b*) *b* = *max a* (*min* (*negmax* (*Nd ts*)) *b*)
⟨*proof*⟩

Not mm, only min and max. Also max in abrs:

theorem shows *abr_negmax2*: *max a* (*min* (*abr t a b*) *b*) = *max a* (*min* (*negmax t*) *b*)
and $a < b \Longrightarrow$ *max a* (*min* (*abrs ts a b*) *b*) = *max a* (*min* (*negmax* (*Nd ts*)) *b*)
⟨*proof*⟩

Relating iteration from right and left

Enables porting *abr* lemmas to *ab_negmax* lemmas, eg correctness.

fun *mirror* :: *'a tree* ⇒ *'a tree* **where**
mirror (*Lf x*) = *Lf x* |
mirror (*Nd ts*) = *Nd* (*rev* (*map mirror ts*))

lemma *abrs_append*:
abrs (*ts1* @ *ts2*) *a b* = (*let m* = *abrs ts2 a b in if* $m \geq b$ *then m else abrs ts1 m b*)
⟨*proof*⟩

lemma *ab_negmax_abr_mirror*:
shows $a<b \Longrightarrow$ *ab_negmax a b t* = *abr* (*mirror t*) *a b*
and $a<b \Longrightarrow$ *ab_negmaxs a b ts* = *abrs* (*rev* (*map mirror ts*)) *a b*
⟨*proof*⟩

lemma *negmax_mirror*:
fixes *t* :: *'a::de_morgan_order tree* **and** *ts* :: *'a::de_morgan_order tree list*
shows *negmax* (*mirror t*) = *negmax t* ∧ *negmax* (*Nd* (*rev* (*map mirror ts*))) = *negmax* (*Nd ts*)
⟨*proof*⟩

Correctness of *ab_negmax* from correctness of *abr*:

theorem *fishburn_ab_negmax_negmax_mirror*:
shows $a < b \Longrightarrow$ *fishburn a b* (*negmax t*) (*ab_negmax a b t*)
and $a < b \Longrightarrow$ *fishburn a b* (*negmax* (*Nd ts*)) (*ab_negmaxs a b ts*)
⟨*proof*⟩

2.5.4 From the Right, Fail-Soft

Starting at ⊥ (after Fishburn)

fun abr' :: $('a::de\_morgan\_order)tree \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **and** $abrs'$ **where**
abr' $(Lf\ x)\ a\ b = x$ |
abr' $(Nd\ ts)\ a\ b = abrs'\ ts\ a\ b$ |

$abrs'\ []\ a\ b = \bot$ |
$abrs'\ (t\#ts)\ a\ b = (let\ m = abrs'\ ts\ a\ b\ in$
$\ \ if\ m \geq b\ then\ m\ else\ max\ (-\ abr'\ t\ (-b)\ (-\ max\ m\ a))\ m)$

lemma $abr01'\_negate$:
shows $abr0'\ (negate\ f\ t)\ a\ b = -\ abr1'\ (negate\ (\neg f)\ t)\ (-a)\ (-b)$
and $abrs0'\ (map\ (negate\ f)\ ts)\ a\ b = -\ abrs1'\ (map\ (negate\ (\neg f))\ ts)\ (-a)\ (-b)$
and $abr1'\ (negate\ f\ t)\ a\ b = -\ abr0'\ (negate\ (\neg f)\ t)\ (-a)\ (-b)$
and $abrs1'\ (map\ (negate\ f)\ ts)\ a\ b = -\ abrs0'\ (map\ (negate\ (\neg f))\ ts)\ (-a)\ (-b)$
⟨*proof*⟩

lemma abr_abr0':
shows $abr'\ t\ a\ b = abr0'\ (negate\ False\ t)\ a\ b$
and $abrs'\ ts\ a\ b = abrs0'\ (map\ (negate\ True)\ ts)\ a\ b$
⟨*proof*⟩

corollary $fishburn\_abr'\_negmax\_cor$:
shows $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ \ \ (abr'\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ (Nd\ ts))\ (abrs'\ ts\ a\ b)$
⟨*proof*⟩

lemma abr'_exact: $[\![\ v = negmax\ t;\ a \leq v \wedge v \leq b]\!] \Longrightarrow abr'\ t\ a\ b = v$
⟨*proof*⟩

Now a lot of copy-paste-modify from *bounded_linorder*.

theorem
shows $a < b \Longrightarrow fishburn\ a\ b\ (abr'\ t\ a\ b)\ (abr\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (abrs'\ ts\ a\ b)\ (abrs\ ts\ a\ b)$
⟨*proof*⟩

theorem $fishburn2\_abr\_abr'$:
shows $a < b \Longrightarrow fishburn\ a\ b\ (abr'\ t\ a\ b)\ (abr\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (abrs'\ ts\ a\ b)\ (abrs\ ts\ a\ b)$
⟨*proof*⟩

theorem $fishburn\_abr'\_negmax$:
shows $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ t)\ \ \ (abr'\ t\ a\ b)$
and $a < b \Longrightarrow fishburn\ a\ b\ (negmax\ (Nd\ ts))\ (abrs'\ ts\ a\ b)$
⟨*proof*⟩

Automatic proof:

theorem

shows $a < b \Longrightarrow$ *fishburn* a b (*negmax* t) (*abr′* t a b)
and $a < b \Longrightarrow$ *fishburn* a b (*negmax* (*Nd ts*)) (*abrs′ ts* a b)
⟨*proof*⟩

Also returning the searched tree

Hard:

fun *abtr* :: (*′a::de_morgan_order*) *tree* ⇒ *′a* ⇒ *′a* ⇒ *′a* ∗ *′a tree* **and** *abtrs* **where**
abtr (*Lf x*) *a b* = (*x*, *Lf x*) |
abtr (*Nd ts*) *a b* = (*let* (*m*,*us*) = *abtrs ts a b in* (*m*, *Nd us*)) |

abtrs [] *a b* = (*a*,[]) |
abtrs (*t*#*ts*) *a b* = (*let* (*m*,*us*) = *abtrs ts a b in*
if m ≥ *b then* (*m*,*us*) *else let* (*n*,*u*) = *abtr t* (−*b*) (−*m*) *in* (*max* (−*n*) *m*,*u*#*us*))

Soft:

fun *abtr′* :: (*′a::de_morgan_order*) *tree* ⇒ *′a* ⇒ *′a* ⇒ *′a* ∗ *′a tree* **and** *abtrs′* **where**
abtr′ (*Lf x*) *a b* = (*x*, *Lf x*) |
abtr′ (*Nd ts*) *a b* = (*let* (*m*,*us*) = *abtrs′ ts a b in* (*m*, *Nd us*)) |

abtrs′ [] *a b* = (⊥,[]) |
abtrs′ (*t*#*ts*) *a b* = (*let* (*m*,*us*) = *abtrs′ ts a b in*
if m ≥ *b then* (*m*,*us*) *else let* (*n*,*u*) = *abtr′ t* (−*b*) (− *max m a*) *in* (*max* (−*n*) *m*,*u*#*us*))

lemma *fst_abtr*:
shows *fst*(*abtr t a b*) = *abr t a b*
and *fst*(*abtrs ts a b*) = *abrs ts a b*
⟨*proof*⟩

lemma *fst_abtr′*:
shows *fst*(*abtr′ t a b*) = *abr′ t a b*
and *fst*(*abtrs′ ts a b*) = *abrs′ ts a b*
⟨*proof*⟩

lemma *snd_abtr′_abtr*:
shows $a < b \Longrightarrow$ *snd*(*abtr′ t a b*) = *snd*(*abtr t a b*)
and $a < b \Longrightarrow$ *snd*(*abtrs′ ts a b*) = *snd*(*abtrs ts a b*)
⟨*proof*⟩

Fail-Soft Generalized

fun *abir′* :: _ ⇒ (*′a::de_morgan_order*)*tree* ⇒ *′a* ⇒ *′a* ⇒ *′a* **and** *abirs′* **where**
abir′ i0 (*Lf x*) *a b* = *x* |
abir′ i0 (*Nd ts*) *a b* = *abirs′ i0* (*i0* (*map* (*negate True*) *ts*) *a*) *ts a b* |

abirs′ i0 i [] *a b* = *i* |
abirs′ i0 i (*t*#*ts*) *a b* =

(*let m* = *abirs′ i0 i ts a b*
in if m $\geq$ *b then m else max* (− *abir′ i0 t* (− *b*) (− *max m a*)) *m*)

abbreviation *neg_all* ≡ *negate True o negate False*

lemma *neg_all_negate*: *neg_all* (*negate f t*) = *negate* (¬*f*) *t*
⟨*proof*⟩

lemma *neg_all_negate′*: *neg_all o negate f* = *negate* (¬*f*)
⟨*proof*⟩

lemma *abir01′_negate*:
shows ∀ *ts a. i1 ts a* = − *i0* (*map neg_all ts*) (−*a*) ⟹
abir0′ i0 i1 (*negate f t*) *a b* = − *abir1′ i0 i1* (*negate* (¬*f*) *t*) (−*a*) (−*b*)
and ∀ *ts a. i1 ts a* = − *i0* (*map neg_all ts*) (−*a*) ⟹
abirs0′ i0 i1 i (*map* (*negate f*) *ts*) *a b* = − *abirs1′ i0 i1* (−*i*) (*map* (*negate* (¬*f*)) *ts*) (−*a*) (−*b*)
and ∀ *ts a. i1 ts a* = − *i0* (*map neg_all ts*) (−*a*) ⟹
abir1′ i0 i1 (*negate f t*) *a b* = − *abir0′ i0 i1* (*negate* (¬*f*) *t*) (−*a*) (−*b*)
and ∀ *ts a. i1 ts a* = − *i0* (*map neg_all ts*) (−*a*) ⟹
abirs1′ i0 i1 i (*map* (*negate f*) *ts*) *a b* = − *abirs0′ i0 i1* (−*i*) (*map* (*negate* (¬*f*)) *ts*) (−*a*) (−*b*)
⟨*proof*⟩

lemma *abir′_abir0′*:
shows *abir′ i0 t a b*
= *abir0′ i0* (λ*ts a.* − *i0* (*map neg_all ts*) (−*a*)) (*negate False t*) *a b*
and *abirs′ i0 i ts a b*
= *abirs0′ i0* (λ*ts a.* − *i0* (*map neg_all ts*) (−*a*)) *i* (*map* (*negate True*) *ts*) *a b*
⟨*proof*⟩

corollary *fishburn_abir′_negmax_cor*:
shows *a* < *b* ⟹ *bnd i0* (λ*ts a.* − *i0* (*map neg_all ts*) (−*a*)) ⟹ *fishburn a b* (*negmax t*) (*abir′ i0 t a b*)
and *a* < *b* ⟹ *bnd i0* (λ*ts a.* − *i0* (*map neg_all ts*) (−*a*)) ⟹ *fishburn a b* (*max i* (*negmax* (*Nd ts*))) (*abirs′ i0 i ts a b*)
⟨*proof*⟩

end

Chapter 3

Distributive Lattices

theory *Alpha_Beta_Lattice*
imports *Alpha_Beta_Linear*
begin

class *distrib_bounded_lattice* = *distrib_lattice* + *bounded_lattice*

instance *bool* :: *distrib_bounded_lattice* ⟨*proof*⟩
instance *ereal* :: *distrib_bounded_lattice* ⟨*proof*⟩
instance *set* :: (*type*) *distrib_bounded_lattice* ⟨*proof*⟩

unbundle *lattice_syntax*

3.1 Game Tree Evaluation

fun *sups* :: (*'a*::*bounded_lattice*) *list* ⇒ *'a* **where**
sups [] = ⊥ |
sups (*x*#*xs*) = *x* ⊔ *sups xs*

fun *infs* :: (*'a*::*bounded_lattice*) *list* ⇒ *'a* **where**
infs [] = ⊤ |
infs (*x*#*xs*) = *x* ⊓ *infs xs*

fun *supinf* :: (*'a*::*distrib_bounded_lattice*) *tree* ⇒ *'a*
and *infsup* :: (*'a*::*distrib_bounded_lattice*) *tree* ⇒ *'a* **where**
supinf (*Lf x*) = *x* |
supinf (*Nd ts*) = *sups* (*map infsup ts*) |
infsup (*Lf x*) = *x* |
infsup (*Nd ts*) = *infs* (*map supinf ts*)

3.2 Distributive Lattices

lemma *sup_inf_assoc*:

$(a::\_::distrib\_lattice) \leq b \Longrightarrow a \sqcup (x \sqcap b) = (a \sqcup x) \sqcap b$
⟨*proof*⟩

lemma *sup_inf_assoc_iff*:
$(a::\_::distrib\_lattice) \sqcup x \sqcap b = a \sqcup y \sqcap b \longleftrightarrow (a \sqcup x) \sqcap b = (a \sqcup y) \sqcap b$
⟨*proof*⟩

Generalization of Knuth and Moore's equivalence modulo:

abbreviation
eq_mod :: ('*a*::*lattice*) ⇒ '*a* ⇒ '*a* ⇒ '*a* ⇒ *bool* (‹(_ ≃/ _/ '(*mod* _,_'))› [51,51,0,0]) **where**
eq_mod x y a b ≡ $a \sqcup x \sqcap b = a \sqcup y \sqcap b$

notation (*latex* **output**) *eq_mod* (‹(_ ≃/ _/ '(mod _,_'))› [51,51,0,0])

ab is bounded by *v* mod *a*,*b*, or the other way around.

abbreviation *bounded* (*a*::_::*lattice*) *b v ab* ≡ $b \sqcap v \leq ab \wedge ab \leq a \sqcup v$

abbreviation *bounded2* :: ('*a*::*lattice*) ⇒ '*a* ⇒ '*a* ⇒ '*a* ⇒ *bool* (‹(_ ⊑/ _/ '(*mod* _,_'))› [51,51,0,0])
where *bounded2 ab v a b* ≡ *bounded a b v ab*

notation (*latex* **output**) *bounded2* (‹(_ ⊑/ _/ '(mod _,_'))› [51,51,0,0])

lemma *bounded_bot_top*:
fixes *v ab* :: '*a*::*distrib_bounded_lattice*
shows *bounded* ⊥ ⊤ *v ab* ⟹ *ab* = *v*
⟨*proof*⟩

bounded implies eq-mod, but not the other way around:

bounded implies eq-mod:

lemma *eq_mod_if_bounded*: **assumes** *bounded a b v ab*
shows $a \sqcup ab \sqcap b = a \sqcup v \sqcap (b::\_::distrib\_lattice)$
⟨*proof*⟩

Converse is not true, even for *linorder*, even if $a < b$:

lemma *let a*=0; *b*=1; *ab*=2; *v*=1
in $a \sqcup ab \sqcap b = a \sqcup v \sqcap (b::nat) \wedge \neg(b \sqcap v \leq ab \wedge ab \leq a \sqcup v)$
⟨*proof*⟩

Because for *linord* we have: *bounded* = *fishburn* ($a < b \Longrightarrow ab \leq v$ (mod a,b) = ($min\ v\ b \leq ab \wedge ab \leq max\ v\ a$)) and *eq_mod* = *knuth* ($a < b \Longrightarrow$ ($max\ a\ (min\ x\ b) = max\ a\ (min\ y\ b)$) = $y \cong x$ (mod a,b)) but we know *fishburn* is stronger than *knuth*.

These equivalences do not even hold as implications in *distrib_lattice*, even if $a < b$. (We need to redefine *knuth* and *fishburn* for *distrib_lattice* first)

context

begin

definition
knuth' $(a::\_::distrib\_lattice)\ b\ x\ y ==$
$((y \le a \longrightarrow x \le a) \wedge (a < y \wedge y < b \longrightarrow y = x) \wedge (b \le y \longrightarrow b \le x))$

lemma $let\ a=\{\};\ b=\{1::int\};\ ab=\{\};\ v=\{0\}$
$in\ \neg\ (a \sqcup ab \sqcap b = a \sqcup v \sqcap b \longrightarrow knuth'\ a\ b\ v\ ab)$
⟨*proof*⟩

lemma $let\ a=\{\};\ b=\{1::int\};\ ab=\{0\};\ v=\{1\}$
$in\ \neg\ (knuth'\ a\ b\ v\ ab \longrightarrow a \sqcup ab \sqcap b = a \sqcup v \sqcap b)$
⟨*proof*⟩

definition
fishburn' $(a::\_::distrib\_lattice)\ b\ v\ ab ==$
$((ab \le a \longrightarrow v \le ab) \wedge (a < ab \wedge ab < b \longrightarrow ab = v) \wedge (b \le ab \longrightarrow ab \le v))$

Same counterexamples as above:

lemma $let\ a=\{\};\ b=\{1::int\};\ ab=\{\};\ v=\{0\}$
$in\ \neg\ (bounded\ a\ b\ v\ ab \longrightarrow fishburn'\ a\ b\ v\ ab)$
⟨*proof*⟩

lemma $let\ a=\{\};\ b=\{1::int\};\ ab=\{0\};\ v=\{1\}$
$in\ \neg\ (fishburn'\ a\ b\ v\ ab \longrightarrow bounded\ a\ b\ v\ ab)$
⟨*proof*⟩

end

3.2.1 Fail-Hard

Basic *ab_sup*

Improved version of Bird and Hughes. No squashing in base case.

fun $ab\_sup :: 'a \Rightarrow 'a \Rightarrow ('a::distrib\_lattice)tree \Rightarrow 'a$ **and** *ab_sups* **and** *ab_inf* **and** *ab_infs* **where**
$ab\_sup\ a\ b\ (Lf\ x) = x\ |$
$ab\_sup\ a\ b\ (Nd\ ts) = ab\_sups\ a\ b\ ts\ |$
$ab\_sups\ a\ b\ [] = a\ |$
$ab\_sups\ a\ b\ (t\#ts) = (let\ a' = a \sqcup ab\_inf\ a\ b\ t\ in\ if\ a' \ge b\ then\ a'\ else\ ab\_sups\ a'\ b\ ts)\ |$
$ab\_inf\ a\ b\ (Lf\ x) = x\ |$
$ab\_inf\ a\ b\ (Nd\ ts) = ab\_infs\ a\ b\ ts\ |$
$ab\_infs\ a\ b\ [] = b\ |$
$ab\_infs\ a\ b\ (t\#ts) = (let\ b' = b \sqcap ab\_sup\ a\ b\ t\ in\ if\ b' \le a\ then\ b'\ else\ ab\_infs\ a\ b'\ ts)$

lemma *ab_sups_ge_a*: $ab\_sups\ a\ b\ ts \ge a$
⟨*proof*⟩

lemma *ab_infs_le_b*: *ab_infs a b ts* $\leq$ *b*
⟨*proof*⟩

lemma *eq_mod_ab_val_auto*:
shows $a \sqcup$ *ab_sup a b t* $\sqcap b = a \sqcup$ *supinf t* $\sqcap b$
and $a \sqcup$ *ab_sups a b ts* $\sqcap b = a \sqcup$ *supinf* (*Nd ts*) $\sqcap b$
and $a \sqcup$ *ab_inf a b t* $\sqcap b = a \sqcup$ *infsup t* $\sqcap b$
and $a \sqcup$ *ab_infs a b ts* $\sqcap b = a \sqcup$ *infsup* (*Nd ts*) $\sqcap b$
⟨*proof*⟩

A readable proof. Some steps still tricky. Complication: sometimes $a \sqcup x \sqcap b$ is better and sometimes $(a \sqcup x) \sqcap b$.

lemma *eq_mod_ab_val*:
shows $a \sqcup$ *ab_sup a b t* $\sqcap b = a \sqcup$ *supinf t* $\sqcap b$
and $a \sqcup$ *ab_sups a b ts* $\sqcap b = a \sqcup$ *supinf* (*Nd ts*) $\sqcap b$
and $a \sqcup$ *ab_inf a b t* $\sqcap b = a \sqcup$ *infsup t* $\sqcap b$
and $a \sqcup$ *ab_infs a b ts* $\sqcap b = a \sqcup$ *infsup* (*Nd ts*) $\sqcap b$
⟨*proof*⟩

corollary *ab_sup_bot_top*: *ab_sup* $\bot$ $\top$ *t* = *supinf t*
⟨*proof*⟩

Predicate *knuth* (and thus *fishburn*) does not hold:

lemma *let a* = {*False*}; *b* = {*False*, *True*}; *t* = *Nd* [*Lf* {*True*}];
ab = *ab_sup a b t*; *v* = *supinf t in v* = {*True*} $\wedge$ *ab* = {*True*,*False*} $\wedge$ *b* $\leq$ *ab* $\wedge$ $\neg$ *b* $\leq$ *v*
⟨*proof*⟩

Worse: *fishburn* (and *knuth*) only caters for a "linear" analysis where *ab* lies wrt $a < b$. But *ab* may not satisfy either of the 3 alternatives in *fishburn*:

lemma *let a* = {}; *b* = {*True*}; *t* = *Nd* [*Lf* {*False*}]; *ab* = *ab_sup a b t*; *v* = *supinf t in*
v = {*False*} $\wedge$ *ab*={*False*} $\wedge$ $\neg$ *ab*$\leq$*a* $\wedge$ $\neg$ *ab*$\geq$*b* $\wedge$ $\neg$ (*a*<*ab* $\wedge$ *ab* < *b*)
⟨*proof*⟩

A stronger correctness property

The stronger correctness property *bounded*:

lemma
shows *bounded a b* (*supinf t*) (*ab_sup a b t*)
and *bounded a b* (*supinf* (*Nd ts*)) (*ab_sups a b ts*)
and *bounded a b* (*infsup t*) (*ab_inf a b t*)
and *bounded a b* (*infsup* (*Nd ts*)) (*ab_infs a b ts*)
⟨*proof*⟩

lemma *bounded_val_ab*:
shows *bounded a b* (*supinf t*) (*ab_sup a b t*)
and *bounded a b* (*supinf* (*Nd ts*)) (*ab_sups a b ts*)

and *bounded a b* (*infsup t*) (*ab_inf a b t*)
and *bounded a b* (*infsup* (*Nd ts*)) (*ab_infs a b ts*)
⟨*proof*⟩

Bird and Hughes

fun *ab_sup*2 :: ('*a*::*distrib_lattice*) ⇒ '*a* ⇒ '*a tree* ⇒ '*a* **and** *ab_sups*2 **and** *ab_inf*2 **and** *ab_infs*2 **where**
*ab_sup*2 *a b* (*Lf x*) = $a \sqcup x \sqcap b$ |
*ab_sup*2 *a b* (*Nd ts*) = *ab_sups*2 *a b ts* |

*ab_sups*2 *a b* [] = *a* |
*ab_sups*2 *a b* (*t*#*ts*) = (*let a'* = *ab_inf*2 *a b t in if a'* = *b then b else ab_sups*2 *a' b ts*) |

*ab_inf*2 *a b* (*Lf x*) = $(a \sqcup x) \sqcap b$ |
*ab_inf*2 *a b* (*Nd ts*) = *ab_infs*2 *a b ts* |

*ab_infs*2 *a b* [] = *b* |
*ab_infs*2 *a b* (*t*#*ts*) = (*let b'* = *ab_sup*2 *a b t in if a* = *b' then a else ab_infs*2 *a b' ts*)

lemma *eq_mod_ab*2*_val*:
shows $a \le b \Longrightarrow$ *ab_sup*2 *a b t* = $a \sqcup$ (*supinf t* $\sqcap$ *b*)
and $a \le b \Longrightarrow$ *ab_sups*2 *a b ts* = $a \sqcup$ (*supinf* (*Nd ts*) $\sqcap$ *b*)
and $a \le b \Longrightarrow$ *ab_inf*2 *a b t* = ($a \sqcup$ *infsup t*) $\sqcap$ *b*
and $a \le b \Longrightarrow$ *ab_infs*2 *a b ts* = ($a \sqcup$ *infsup*(*Nd ts*)) $\sqcap$ *b*
⟨*proof*⟩

corollary *ab_sup*2*_bot_top*: *ab_sup*2 ⊥ ⊤ *t* = *supinf t*
⟨*proof*⟩

Simpler proof with sets; not really surprising.

lemma *ab_sup*2*_bounded_set*:
shows $a \le$(*b*:: _ *set*) $\Longrightarrow$ *ab_sup*2 *a b t* = $a \sqcup$ (*supinf t* $\sqcap$ *b*)
and $a \le b \Longrightarrow$ *ab_sups*2 *a b ts* = $a \sqcup$ (*supinf* (*Nd ts*) $\sqcap$ *b*)
and $a \le b \Longrightarrow$ *ab_inf*2 *a b t* = ($a \sqcup$ *infsup t*) $\sqcap$ *b*
and $a \le b \Longrightarrow$ *ab_infs*2 *a b ts* = ($a \sqcup$ *infsup*(*Nd ts*)) $\sqcap$ *b*
⟨*proof*⟩

Delayed Test

fun *ab_sup*3 :: ('*a*::*distrib_lattice*) ⇒ '*a* ⇒ '*a tree* ⇒ '*a* **and** *ab_sups*3 **and** *ab_inf*3 **and** *ab_infs*3 **where**
*ab_sup*3 *a b* (*Lf x*) = *x* |
*ab_sup*3 *a b* (*Nd ts*) = *ab_sups*3 *a b ts* |

*ab_sups*3 *a b* [] = *a* |
*ab_sups*3 *a b* (*t*#*ts*) = (*if a* ≥ *b then a else ab_sups*3 ($a \sqcup$ *ab_inf*3 *a b t*) *b ts*) |

ab_inf3 a b (Lf x) = x |
ab_inf3 a b (Nd ts) = ab_infs3 a b ts |

ab_infs3 a b [] = b |
ab_infs3 a b (t#ts) = (if a ≥ b then b else ab_infs3 a (b ⊓ ab_sup3 a b t) ts)

lemma *ab_sups3_ge_a*: *ab_sups3 a b ts ≥ a*
⟨*proof*⟩

lemma *ab_infs3_le_b*: *ab_infs3 a b ts ≤ b*
⟨*proof*⟩

lemma *ab_sup3_ab_sup*:
shows *a<b ⟹ ab_sup3 a b t = ab_sup a b t*
and *a<b ⟹ ab_sups3 a b ts = ab_sups a b ts*
and *a<b ⟹ ab_inf3 a b t = ab_inf a b t*
and *a<b ⟹ ab_infs3 a b ts = ab_infs a b ts*
quickcheck[*expect=no_counterexample*]
⟨*proof*⟩

Bird and Hughes plus delayed test

fun *ab_sup4* :: (*'a::distrib_lattice*) ⇒ *'a* ⇒ *'a tree* ⇒ *'a* **and** *ab_sups4* **and** *ab_inf4* **and** *ab_infs4* **where**
ab_sup4 a b (Lf x) = a ⊔ x ⊓ b |
ab_sup4 a b (Nd ts) = ab_sups4 a b ts |

ab_sups4 a b [] = a |
ab_sups4 a b (t#ts) = (if a = b then a else ab_sups4 (ab_inf4 a b t) b ts) |

ab_inf4 a b (Lf x) = (a ⊔ x) ⊓ b |
ab_inf4 a b (Nd ts) = ab_infs4 a b ts |

ab_infs4 a b [] = b |
ab_infs4 a b (t#ts) = (if a = b then b else ab_infs4 a (ab_sup4 a b t) ts)

lemma *ab_sup4_bounded*:
shows *a≤b ⟹ ab_sup4 a b t = a ⊔ (supinf t ⊓ b)*
and *a≤b ⟹ ab_sups4 a b ts = a ⊔ (supinf (Nd ts) ⊓ b)*
and *a≤b ⟹ ab_inf4 a b t = (a ⊔ infsup t) ⊓ b*
and *a≤b ⟹ ab_infs4 a b ts = (a ⊔ infsup(Nd ts)) ⊓ b*
⟨*proof*⟩

lemma *ab_sup4_bounded_set*:
shows *a≤(b:: _ set) ⟹ ab_sup4 a b t = a ⊔ (supinf t ⊓ b)*
and *a≤b ⟹ ab_sups4 a b ts = a ⊔ (supinf (Nd ts) ⊓ b)*

and $a \le b \Longrightarrow ab\_inf4\ a\ b\ t = (a \sqcup infsup\ t) \sqcap b$
and $a \le b \Longrightarrow ab\_infs4\ a\ b\ ts = (a \sqcup infsup(Nd\ ts)) \sqcap b$
⟨*proof*⟩

3.2.2 Fail-Soft

fun $ab\_sup' :: 'a::distrib\_bounded\_lattice \Rightarrow 'a \Rightarrow 'a\ tree \Rightarrow 'a$ **and** ab_sups' ab_inf' ab_infs' **where**
$ab\_sup'\ a\ b\ (Lf\ x) = x$ |
$ab\_sup'\ a\ b\ (Nd\ ts) = ab\_sups'\ a\ b\ \bot\ ts$ |

$ab\_sups'\ a\ b\ m\ [] = m$ |
$ab\_sups'\ a\ b\ m\ (t\#ts) =$
$(let\ m' = m \sqcup (ab\_inf'\ (m \sqcup a)\ b\ t)\ in\ if\ m' \ge b\ then\ m'\ else\ ab\_sups'\ a\ b\ m'\ ts)$ |

$ab\_inf'\ a\ b\ (Lf\ x) = x$ |
$ab\_inf'\ a\ b\ (Nd\ ts) = ab\_infs'\ a\ b\ \top\ ts$ |

$ab\_infs'\ a\ b\ m\ [] = m$ |
$ab\_infs'\ a\ b\ m\ (t\#ts) =$
$(let\ m' = m \sqcap (ab\_sup'\ a\ (m \sqcap b)\ t)\ in\ if\ m' \le a\ then\ m'\ else\ ab\_infs'\ a\ b\ m'\ ts)$

lemma $ab\_sups'\_ge\_m$: $ab\_sups'\ a\ b\ m\ ts \ge m$
⟨*proof*⟩

lemma $ab\_infs'\_le\_m$: $ab\_infs'\ a\ b\ m\ ts \le m$
⟨*proof*⟩

Fail-soft correctness:

lemma $bounded\_val\_ab'$:
shows $bounded\ (a)\ b\ (supinf\ t)\ (ab\_sup'\ a\ b\ t)$
and $bounded\ (a \sqcup m)\ b\ (supinf\ (Nd\ ts))\ (ab\_sups'\ a\ b\ m\ ts)$
and $bounded\ a\ b\ (infsup\ t)\ (ab\_inf'\ a\ b\ t)$
and $bounded\ a\ (b \sqcap m)\ (infsup\ (Nd\ ts))\ (ab\_infs'\ a\ b\ m\ ts)$
⟨*proof*⟩

corollary $ab\_sup'\ \bot\ \top\ t = supinf\ t$
⟨*proof*⟩

lemma $eq\_mod\_ab'\_val$:
shows $a \sqcup ab\_sup'\ a\ b\ t \sqcap b = a \sqcup supinf\ t \sqcap b$
and $(m \sqcup a) \sqcup ab\_sups'\ a\ b\ m\ ts \sqcap b = (m \sqcup a) \sqcup supinf\ (Nd\ ts) \sqcap b$
and $a \sqcup ab\_inf'\ a\ b\ t \sqcap b = a \sqcup infsup\ t \sqcap b$
and $a \sqcup ab\_infs'\ a\ b\ m\ ts \sqcap (m \sqcap b) = a \sqcup infsup\ (Nd\ ts) \sqcap (m \sqcap b)$
⟨*proof*⟩

lemma *ab_sups'_le_ab_sups*: *ab_sups' a b c t* $\sqcap$ *b* $\leq$ *ab_sups* $(a \sqcup c)$ *b t*
⟨*proof*⟩

lemma *ab_sup'_le_ab_sup*: *ab_sup' a b t* $\sqcap$ *b* $\leq$ *ab_sup a b t*
⟨*proof*⟩

Towards *bounded* **of Fail-Soft**

theorem *bounded_ab'_ab*:
  shows *bounded* (*a*) *b* (*ab_sup' a b t*)   (*ab_sup a b t*)
   and *bounded a b* (*ab_sups' a b m ts*) (*ab_sups* (*sup m a*) *b ts*)
   and *bounded a b* (*ab_inf' a b t*)   (*ab_inf a b t*)
   and *bounded a b* (*ab_infs' a b m ts*) (*ab_infs a* (*inf m b*) *ts*)
⟨*proof*⟩

3.3 De Morgan Algebras

Now: also negation. But still not a boolean algebra but only a De Morgan algebra:

class *de_morgan_algebra* = *distrib_bounded_lattice* + *uminus*
opening *lattice_syntax* +
assumes *de_Morgan_inf*: $- (x \sqcap y) = - x \sqcup - y$
assumes *neg_neg*[*simp*]: $- (- x) = x$
begin

lemma *de_Morgan_sup*: $- (x \sqcup y) = - x \sqcap - y$
⟨*proof*⟩

lemma *neg_top*[*simp*]: $- \top = \bot$
⟨*proof*⟩

lemma *neg_bot*[*simp*]: $- \bot = \top$
⟨*proof*⟩

lemma *uminus_eq_iff*[*simp*]: $-a = -b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *uminus_le_reorder*: $(- a \leq b) = (- b \leq a)$
⟨*proof*⟩

lemma *uminus_less_reorder*: $(- a < b) = (- b < a)$
⟨*proof*⟩

lemma *minus_le_minus*[*simp*]: $- a \leq - b \longleftrightarrow b \leq a$
⟨*proof*⟩

lemma *minus_less_minus*[*simp*]: $-\ a < -\ b \longleftrightarrow b < a$
⟨*proof*⟩

lemma *less_uminus_reorder*: $a < -\ b \longleftrightarrow b < -\ a$
⟨*proof*⟩

end

instantiation *ereal* :: *de_morgan_algebra*
begin

instance
⟨*proof*⟩

end

instantiation *set* :: (*type*)*de_morgan_algebra*
begin

instance
⟨*proof*⟩

end

fun *negsup* :: ('*a* :: *de_morgan_algebra*)*tree* ⇒ '*a* **where**
negsup (*Lf x*) = *x* |
negsup (*Nd ts*) = *sups* (*map* (λ*t*. − *negsup t*) *ts*)

fun *negate* :: *bool* ⇒ ('*a*::*de_morgan_algebra*) *tree* ⇒ '*a tree* **where**
negate b (*Lf x*) = *Lf* (*if b then* −*x else x*) |
negate b (*Nd ts*) = *Nd* (*map* (*negate* (¬*b*)) *ts*)

lemma *negate_negate*: *negate f* (*negate f t*) = *t*
⟨*proof*⟩

lemma *uminus_infs*:
  fixes *f* :: '*a* ⇒ '*b*::*de_morgan_algebra*
shows − *infs* (*map f xs*) = *sups* (*map* (λ*x*. − *f x*) *xs*)
⟨*proof*⟩

lemma *supinf_negate*: *supinf* (*negate b t*) = − *infsup* (*negate* (¬*b*) (*t*::(_::*de_morgan_algebra*)*tree*))
⟨*proof*⟩

lemma *negsup_supinf_negate*: *negsup t* = *supinf*(*negate False t*)
⟨*proof*⟩

3.3.1 Fail-Hard

fun *ab_negsup* :: $'a \Rightarrow 'a \Rightarrow ('a{::}de\_morgan\_algebra)tree \Rightarrow 'a$ **and** *ab_negsups*
where
$ab\_negsup\ a\ b\ (Lf\ x) = x\ |$
$ab\_negsup\ a\ b\ (Nd\ ts) = ab\_negsups\ a\ b\ ts\ |$

$ab\_negsups\ a\ b\ [] = a\ |$
$ab\_negsups\ a\ b\ (t\#ts) =$
$(let\ a' = a \sqcup -\ ab\_negsup\ (-b)\ (-a)\ t$
$in\ if\ a' \geq b\ then\ a'\ else\ ab\_negsups\ a'\ b\ ts)$

A direct *bounded* proof:

lemma *ab_negsups_ge_a*: $ab\_negsups\ a\ b\ ts \geq a$
⟨*proof*⟩

lemma *bounded_val_ab_neg*:
shows $bounded\ (a)\ b\ (negsup\ t)\ (ab\_negsup\ (a)\ b\ t)$
and $bounded\ a\ b\ (negsup\ (Nd\ ts))\ (ab\_negsups\ (a)\ b\ ts)$
⟨*proof*⟩

An indirect proof:

theorem *ab_sup_ab_negsup*:
shows $ab\_sup\ a\ b\ t = ab\_negsup\ a\ b\ (negate\ False\ t)$
and $ab\_sups\ a\ b\ ts = ab\_negsups\ a\ b\ (map\ (negate\ True)\ ts)$
and $ab\_inf\ a\ b\ t = -\ ab\_negsup\ (-b)\ (-a)\ (negate\ True\ t)$
and $ab\_infs\ a\ b\ ts = -\ ab\_negsups\ (-b)\ (-a)\ (map\ (negate\ False)\ ts)$
⟨*proof*⟩

corollary *ab_negsup_bot_top*: $ab\_negsup\ \bot\ \top\ t = negsup\ t$
⟨*proof*⟩

Correctness statements derived from non-negative versions:

corollary *eq_mod_ab_negsup_supinf_negate*:
$a \sqcup ab\_negsup\ a\ b\ t \sqcap b = a \sqcup supinf\ (negate\ False\ t) \sqcap b$
⟨*proof*⟩

corollary *bounded_negsup_ab_negsup*:
$bounded\ a\ b\ (negsup\ t)\ (ab\_negsup\ a\ b\ t)$
⟨*proof*⟩

3.3.2 Fail-Soft

fun *ab_negsup'* :: $'a \Rightarrow 'a \Rightarrow ('a{::}de\_morgan\_algebra)tree \Rightarrow 'a$ **and** *ab_negsups'*
where
$ab\_negsup'\ a\ b\ (Lf\ x) = x\ |$
$ab\_negsup'\ a\ b\ (Nd\ ts) = (ab\_negsups'\ a\ b\ \bot\ ts)\ |$

$ab\_negsups'\ a\ b\ m\ [] = m\ |$

ab_negsups' a b m (t#ts) = (let m' = sup m (− ab_negsup' (−b) (− sup m a) t) in
if m' ≥ b then m' else ab_negsups' a b m' ts)

Relate un-negated to negated:

theorem *ab_sup'_ab_negsup'*:
shows *ab_sup' a b t = ab_negsup' a b (negate False t)*
and *ab_sups' a b m ts = ab_negsups' a b m (map (negate True) ts)*
and *ab_inf' a b t = − ab_negsup' (−b) (−a) (negate True t)*
and *ab_infs' a b m ts = − ab_negsups' (−b) (−a) (−m) (map (negate False) ts)*
⟨*proof*⟩

lemma *ab_negsups'_ge_a*: *ab_negsups' a b m ts ≥ m*
⟨*proof*⟩

theorem *bounded_val_ab'_neg*:
shows *bounded a b (negsup t) (ab_negsup' a b t)*
and *bounded (sup a m) b (negsup (Nd ts)) (ab_negsups' a b m ts)*
⟨*proof*⟩

corollary *bounded a b (negsup t) (ab_negsup' a b t)*
⟨*proof*⟩

theorem *bounded_ab_neg'_ab_neg*:
shows *bounded a b (ab_negsup' a b t) (ab_negsup a b t)*
and *bounded (sup a m) b (ab_negsups' a b m ts) (ab_negsup (a ⊔ m) b (Nd ts))*
⟨*proof*⟩

end

Chapter 4

An Application: Tic-Tac-Toe

theory *TicTacToe*
imports
Alpha_Beta_Pruning.Alpha_Beta_Linear
begin

We formalize a general nxn version of tic-tac-toe (noughts and crosses). The winning condition is very simple: a full horizontal, vertical or diagonal line occupied by one player.

A square is either empty (*None*) or occupied by one of the two players (*Some b*).

type_synonym *sq* = *bool option*
type_synonym *row* = *sq list*
type_synonym *position* = *row list*

Successor positions:

fun *next_rows* :: *sq* ⇒ *row* ⇒ *row list* **where**
next_rows s′ (s#ss) = (if s=None then [s′#ss] else []) @ map ((#) s) (next_rows s′ ss) |
next_rows _ [] = []

fun *next_poss* :: *sq* ⇒ *position* ⇒ *position list* **where**
next_poss s′ (ss#sss) = map (λss′. ss′ # sss) (next_rows s′ ss) @ map ((#) ss) (next_poss s′ sss) |
next_poss _ [] = []

A game is won if a full line is occupied by a given square:

fun *diag* :: *′a list list* ⇒ *′a list* **where**
diag ((x#_) # xss) = x # diag (map tl xss) |
diag [] = []

fun *lines* :: *position* ⇒ *sq list list* **where**
lines sss = diag sss # diag (map rev sss) # sss @ transpose sss

fun *won* :: *sq* ⇒ *position* ⇒ *bool* **where**

won sq pos = ($\exists$ *ss* $\in$ *set* (*lines pos*). $\forall$ *s* $\in$ *set ss*. *s* = *sq*)

How many lines are almost won (i.e. all *sq* except one *None*)? Not actually used for heuristic evaluation, too slow.

fun *awon* :: *sq* $\Rightarrow$ *position* $\Rightarrow$ *nat* **where**
awon sq sss = *length* (*filter* (λ*ss*. *filter* (λ*s*. *s*$\neq$*sq*) *ss* = [*None*]) (*lines sss*))

The game tree up to a given depth *n*. Trees at depth $\geq n$ are replaced by *Lf* 0 for simplicity; no heuristic evaluation.

fun *tree* :: *nat* $\Rightarrow$ *bool* $\Rightarrow$ *position* $\Rightarrow$ *ereal tree* **where**
tree (*Suc n*) *b pos* = (
  if won (*Some* ($\neg$*b*)) *pos then Lf*(*if b then* $-\infty$ *else* ∞) — Opponent won
  else
  case next_poss (*Some b*) *pos of*
    [] $\Rightarrow$ *Lf* 0 — Draw |
    poss $\Rightarrow$ *Nd* (*map* (*tree n* ($\neg$*b*)) *poss*)) |
tree 0 *b pos* = *Lf* 0

definition *start* :: *nat* $\Rightarrow$ *position* **where**
start n = *replicate n* (*replicate n None*)

Now we evaluate the game for small *n*.

The trivial cases:

lemma *maxmin* (*tree* 2 *True* (*start* 1)) = ∞
⟨*proof*⟩

lemma *maxmin* (*tree* 5 *True* (*start* 2)) = ∞
⟨*proof*⟩

3x3, full game tree (depth=10), no noticeable speedup of alpha-beta.

lemma *maxmin* (*tree* 10 *True* (*start* 3)) = 0
⟨*proof*⟩
lemma *ab_max* ($-\infty$) ∞ (*tree* 10 *True* (*start* 3)) = 0
⟨*proof*⟩

4x4, game tree up to depth 7, alpha-beta noticeably faster.

lemma *maxmin* (*tree* 7 *True* (*start* 4)) = 0
⟨*proof*⟩
lemma *ab_max* ($-\infty$) ∞ (*tree* 7 *True* (*start* 4)) = 0
⟨*proof*⟩

end