

Simplicial complexes and Boolean functions*

Jesús María Aransay Azofra, Alejandro del Campo López, Julius Michaelis

December 14, 2021

Abstract

In this work we formalise the isomorphism between simplicial complexes of dimension n and monotone Boolean functions in n variables, mainly following the definitions and results as introduced by N. A. Scoville [3, Ch. 6]. We also take advantage of the AFP representation of ROBDD (Reduced Ordered Binary Decision Diagrams) [2] to compute the ROBDD representation of a given simplicial complex (by means of the isomorphism to Boolean functions). Some examples of simplicial complexes and associated Boolean functions are also presented.

Contents

1	Introduction	2
2	Boolean functions	2
3	Threshold function	3
4	Simplicial Complexes	4
5	Simplicial complex induced by a monotone Boolean function	6
6	The simplicial complex induced by the threshold function	8
7	Bijection between simplicial complexes and monotone Boolean functions	15
8	Converting boolean functions to BDTs	17
9	Relation between type $bool\ vec \Rightarrow bool$ and type $'a\ boolfunc$	18
10	Definition of <i>evasive</i> Boolean function	20

*This research was partially funded by Ministerio de Ciencia e Innovación (Spain), grant number PID2020-116641GB-I00.

11 Detour: Lexicographic ordering for lists	21
12 Executably converting Simplicial Complexes to BDDs	22
13 Binary operations over Boolean functions and simplicial complexes	29

1 Introduction

```

theory Boolean-functions
  imports
    Main
    Jordan-Normal-Form.Matrix
begin

```

2 Boolean functions

Definition of monotonicity

We consider (monotone) Boolean functions over vectors of length n , so that we can later prove that those are isomorphic to simplicial complexes of dimension n (in n vertexes).

```

locale boolean-functions
  = fixes  $n::nat$ 
begin

```

```

definition bool-fun-dim-n :: (bool vec => bool) set
  where bool-fun-dim-n = { $f. f \in \text{carrier-vec } n \rightarrow (UNIV::\text{bool set})$ }

```

```

definition monotone-bool-fun :: (bool vec => bool) => bool
  where monotone-bool-fun  $f \equiv (\text{mono-on } f (\text{carrier-vec } n))$ 

```

```

definition monotone-bool-fun-set :: (bool vec => bool) set
  where monotone-bool-fun-set = (Collect monotone-bool-fun)

```

Some examples of Boolean functions

```

definition bool-fun-top :: bool vec => bool
  where bool-fun-top  $f = \text{True}$ 

```

```

definition bool-fun-bot :: bool vec => bool
  where bool-fun-bot  $f = \text{False}$ 

```

```

end

```

3 Threshold function

definition *count-true* :: *bool vec* => *nat*
where *count-true v* = *sum* ($\lambda i. \text{if } \text{vec-index } v \ i \ \text{then } 1 \ \text{else } 0 :: \text{nat}$) {*0..<dim-vec v*}

lemma *vec-index* (*vec* (5::*nat*) ($\lambda i. \text{False}$)) 2 = *False*
(*proof*)

lemma *vec-index* (*vec* (5::*nat*) ($\lambda i. \text{True}$)) 3 = *True*
(*proof*)

lemma *count-true* (*vec* (1::*nat*) ($\lambda i. \text{True}$)) = 1
(*proof*)

lemma *count-true* (*vec* (2::*nat*) ($\lambda i. \text{True}$)) = 2
(*proof*)

lemma *count-true* (*vec* (5::*nat*) ($\lambda i. \text{True}$)) = 5
(*proof*)

The threshold function is a Boolean function which also satisfies the condition of being *evasive*. We follow the definition by Scoville [3, Problem 6.5].

definition *bool-fun-threshold* :: *nat* => (*bool vec* => *bool*)
where *bool-fun-threshold i* = ($\lambda v. \text{if } i \leq \text{count-true } v \ \text{then } \text{True} \ \text{else } \text{False}$)

context *boolean-functions*
begin

lemma *mono-on bool-fun-top UNIV*
(*proof*)

lemma *monotone-bool-fun bool-fun-top*
(*proof*)

lemma *mono-on bool-fun-bot UNIV*
(*proof*)

lemma *monotone-bool-fun bool-fun-bot*
(*proof*)

lemma
monotone-count-true:
assumes *ulev*: (*u*::*bool vec*) \leq *v*
shows *count-true u* \leq *count-true v*
(*proof*)

The threshold function is monotone.

lemma
monotone-threshold:
assumes $ulev: (u::bool\ vec) \leq v$
shows $bool_fun_threshold\ n\ u \leq bool_fun_threshold\ n\ v$
 $\langle proof \rangle$

lemma
assumes $(u::bool\ vec) \leq v$
and $n < dim_vec\ u$
shows $bool_fun_threshold\ n\ u \leq bool_fun_threshold\ n\ v$
 $\langle proof \rangle$

lemma *mono-on* $(bool_fun_threshold\ n)\ UNIV$
 $\langle proof \rangle$

lemma *monotone-bool-fun* $(bool_fun_threshold\ n)$
 $\langle proof \rangle$

end

end

theory *Simplicial-complex*
imports
Boolean-functions
begin

4 Simplicial Complexes

lemma *Pow-singleton*: $Pow\ \{a\} = \{\{\},\{a\}\}$ $\langle proof \rangle$

lemma *Pow-pair*: $Pow\ \{a,b\} = \{\{\},\{a\},\{b\},\{a,b\}\}$ $\langle proof \rangle$

locale *simplicial-complex*
= **fixes** $n::nat$
begin

A simplex (in n vertexes) is any set of vertexes, including the empty set.

definition *simplices* $:: nat\ set\ set$
where $simplices = Pow\ \{0..<n\}$

lemma $\{\} \in simplices$
 $\langle proof \rangle$

lemma $\{0..<n\} \in simplices$
 $\langle proof \rangle$

lemma *finite-simplex*:
assumes $\sigma \in simplices$

shows *finite* σ
<proof>

A simplicial complex (in n vertexes) is a collection of sets of vertexes such that every subset of a set of vertexes also belongs to the simplicial complex.

definition *simplicial-complex* :: *nat set set => bool*
where *simplicial-complex* $K \equiv (\forall \sigma \in K. (\sigma \in \text{simplices}) \wedge (\text{Pow } \sigma) \subseteq K)$

lemma
finite-simplicial-complex:
assumes *simplicial-complex* K
shows *finite* K
<proof>

lemma *finite-simplices*:
assumes *simplicial-complex* K
and $v \in K$
shows *finite* v
<proof>

definition *simplicial-complex-set* :: *nat set set set*
where *simplicial-complex-set* = *(Collect simplicial-complex)*

lemma *simplicial-complex-empty-set*:
fixes $K::\text{nat set set}$
assumes $k: \text{simplicial-complex } K$
shows $K = \{\} \vee \{\} \in K$ *<proof>*

lemma
simplicial-complex-monotone:
fixes $K::\text{nat set set}$
assumes $k: \text{simplicial-complex } K$ **and** $s: s \in K$ **and** $rs: r \subseteq s$
shows $r \in K$
<proof>

One example of simplicial complex with four simplices.

lemma
assumes *three*: $(3::\text{nat}) < n$
shows *simplicial-complex* $\{\{\}, \{0\}, \{1\}, \{2\}, \{3\}\}$
<proof>

lemma \neg *simplicial-complex* $\{\{0,1\}, \{1\}\}$
<proof>

Another example of simplicial complex with five simplices.

lemma
assumes *three*: $(3::\text{nat}) < n$
shows *simplicial-complex* $\{\{\}, \{0\}, \{1\}, \{2\}, \{3\}, \{0,1\}\}$

<proof>

Another example of simplicial complex with ten simplices.

lemma

assumes *three*: ($\mathcal{I}::\text{nat}$) $< n$

shows *simplicial-complex*

$\{\{2,\mathcal{I}\},\{1,\mathcal{I}\},\{1,2\},\{0,\mathcal{I}\},\{0,2\},\{\mathcal{I}\},\{2\},\{1\},\{0\},\{\}\}$

<proof>

end

5 Simplicial complex induced by a monotone Boolean function

In this section we introduce the definition of the simplicial complex induced by a monotone Boolean function, following the definition in Scoville [3, Def. 6.9].

First we introduce the set of tuples for which a Boolean function is *False*.

definition *ceros-of-boolean-input* :: $\text{bool vec} \Rightarrow \text{nat set}$

where *ceros-of-boolean-input* $v = \{x. x < \text{dim-vec } v \wedge \text{vec-index } v \ x = \text{False}\}$

lemma

ceros-of-boolean-input-l-dim:

assumes $a: a \in \text{ceros-of-boolean-input } v$

shows $a < \text{dim-vec } v$

<proof>

lemma *ceros-of-boolean-input* $v = \{x. x < \text{dim-vec } v \wedge \neg \text{vec-index } v \ x\}$

<proof>

lemma

ceros-of-boolean-input-complementary:

shows *ceros-of-boolean-input* $v = \{x. x < \text{dim-vec } v\} - \{x. \text{vec-index } v \ x\}$

<proof>

lemma *monotone-ceros-of-boolean-input*:

fixes r **and** $s::\text{bool vec}$

assumes $r\text{-le-}s: r \leq s$

shows *ceros-of-boolean-input* $s \subseteq \text{ceros-of-boolean-input } r$

<proof>

We introduce here instantiations of the *typbool* type for the type classes *classzero* and *classone* that will simplify notation at some points:

instantiation *bool* :: $\{\text{zero},\text{one}\}$

begin

definition

zero-bool-def: $0 == \text{False}$

definition

one-bool-def: $1 == \text{True}$

instance $\langle \text{proof} \rangle$

end

Definition of the simplicial complex induced by a Boolean function f in dimension n .

definition

simplicial-complex-induced-by-monotone-boolean-function

$:: \text{nat} \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \Rightarrow \text{nat set set}$

where *simplicial-complex-induced-by-monotone-boolean-function* $n f =$

$\{y. \exists x. \text{dim-vec } x = n \wedge f x \wedge \text{zeros-of-boolean-input } x = y\}$

The simplicial complex induced by a Boolean function is a subset of the powerset of the set of vertexes.

lemma

simplicial-complex-induced-by-monotone-boolean-function-subset:

simplicial-complex-induced-by-monotone-boolean-function $n (v::\text{bool vec} \Rightarrow \text{bool})$

$\subseteq \text{Pow } (\{0..n\}::\text{nat set})$

$\langle \text{proof} \rangle$

corollary

simplicial-complex-induced-by-monotone-boolean-function $n (v::\text{bool vec} \Rightarrow \text{bool})$

$\subseteq \text{Pow } (\text{UNIV}::\text{nat set}) \langle \text{proof} \rangle$

The simplicial complex induced by a monotone Boolean function is a simplicial complex. This result is proven in Scoville as part of the proof of Proposition 6.16 [3, Prop. 6.16].

context *simplicial-complex*

begin

lemma

monotone-bool-fun-induces-simplicial-complex:

assumes *mon*: *boolean-functions.monotone-bool-fun* $n f$

shows *simplicial-complex* (*simplicial-complex-induced-by-monotone-boolean-function* $n f$)

$\langle \text{proof} \rangle$

end

Example 6.10 in Scoville, the threshold function for 2 in dimension 4 (with vertexes 0,1,2,3)

definition *bool-fun-threshold-2-3* :: *bool vec => bool*
where *bool-fun-threshold-2-3* = ($\lambda v. \text{if } 2 \leq \text{count-true } v \text{ then True else False}$)

lemma *set-list-four*: **shows** $\{0..<4\} = \text{set } [0,1,2,3::\text{nat}]$ *<proof>*

lemma *comp-fun-commute-lambda*:
comp-fun-commute-on UNIV ((+)
 $\circ (\lambda i. \text{if } \text{vec } 4 \text{ f } \$ i \text{ then } 1 \text{ else } (0::\text{nat}))$)
<proof>

lemma *bool-fun-threshold-2-3*
 $(\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False})) = \text{True}$
<proof>

lemma
 $0 \notin \text{ceros-of-boolean-input } (\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False}))$
and $1 \notin \text{ceros-of-boolean-input } (\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False}))$
and $2 \in \text{ceros-of-boolean-input } (\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False}))$
and $3 \in \text{ceros-of-boolean-input } (\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False}))$
and $\{2,3\} \subseteq \text{ceros-of-boolean-input } (\text{vec } 4 (\lambda i. \text{if } i = 0 \vee i = 1 \text{ then True else False}))$
<proof>

lemma *bool-fun-threshold-2-3* $(\text{vec } 4 (\lambda i. \text{if } i = 3 \text{ then True else False})) = \text{False}$
<proof>

lemma *bool-fun-threshold-2-3* $(\text{vec } 4 (\lambda i. \text{if } i = 0 \text{ then False else True}))$
<proof>

6 The simplicial complex induced by the threshold function

lemma
empty-set-in-simplicial-complex-induced:
 $\{\} \in \text{simplicial-complex-induced-by-monotone-boolean-function } 4 \text{ bool-fun-threshold-2-3}$
<proof>

lemma *singleton-in-simplicial-complex-induced*:
assumes $x: x < 4$
shows $\{x\} \in \text{simplicial-complex-induced-by-monotone-boolean-function } 4 \text{ bool-fun-threshold-2-3}$
(is $?A \in \text{simplicial-complex-induced-by-monotone-boolean-function } 4 \text{ bool-fun-threshold-2-3}$)
<proof>

lemma *pair-in-simplicial-complex-induced*:
assumes $x: x < 4$ **and** $y: y < 4$
shows $\{x,y\} \in \text{simplicial-complex-induced-by-monotone-boolean-function } 4 \text{ bool-fun-threshold-2-3}$
(is $?A \in \text{simplicial-complex-induced-by-monotone-boolean-function } 4 \text{ bool-fun-threshold-2-3}$)
<proof>

lemma *finite-False*: $\text{finite } \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{False}\}$
 ⟨proof⟩

lemma *finite-True*: $\text{finite } \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{True}\}$
 ⟨proof⟩

lemma *UNIV-disjoint*: $\{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{True}\}$
 $\cap \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{False}\} = \{\}$
 ⟨proof⟩

lemma *UNIV-union*: $\{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{True}\}$
 $\cup \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{False}\} = \{x. x < \text{dim-vec } a\}$
 ⟨proof⟩

lemma *card-UNIV-union*:
 $\text{card } \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{True}\}$
 $+ \text{card } \{x. x < \text{dim-vec } a \wedge \text{vec-index } (a::\text{bool vec}) x = \text{False}\}$
 $= \text{card } \{x. x < \text{dim-vec } a\}$
 (is card ?true + card ?false = -)
 ⟨proof⟩

lemma *card-complementary*:
 $\text{card } (\text{ceros-of-boolean-input } v)$
 $+ \text{card } \{x. x < (\text{dim-vec } v) \wedge (\text{vec-index } v x = \text{True})\} = (\text{dim-vec } v)$
 ⟨proof⟩

corollary
card-ceros-of-boolean-input:
shows $\text{card } (\text{ceros-of-boolean-input } a) \leq \text{dim-vec } a$
 ⟨proof⟩

lemma
vec-fun:
assumes $v \in \text{carrier-vec } n$
shows $\exists f. v = \text{vec } n f$ ⟨proof⟩

corollary
assumes $\text{dim-vec } v = n$
shows $\exists f. v = \text{vec } n f$
 ⟨proof⟩

lemma
vec-l-eq:
assumes $i < n$
shows $\text{vec } (\text{Suc } n) f \$ i = \text{vec } n f \$ i$
 ⟨proof⟩

lemma

card-boolean-function:
assumes $d: v \in \text{carrier-vec } n$
shows $\text{card } \{x. x < n \wedge v \$ x = \text{True}\} = (\sum i = 0..<n. \text{if } v \$ i \text{ then } 1 \text{ else } 0::\text{nat})$
 $\langle \text{proof} \rangle$

lemma *card-ceros-count-UNIV:*
shows $\text{card } (\text{ceros-of-boolean-input } a) + \text{count-true } ((a::\text{bool vec})) = \text{dim-vec } a$
 $\langle \text{proof} \rangle$

We calculate the carrier set of the *ceros-of-boolean-input* function for dimensions 2, 3 and 4.

Vectors of dimension 2.

lemma
dim-vec-2-cases:
assumes $dx: \text{dim-vec } x = 2$
shows $(x \$ 0 = x \$ 1 = \text{True}) \vee (x \$ 0 = \text{False} \wedge x \$ 1 = \text{True})$
 $\vee (x \$ 0 = \text{True} \wedge x \$ 1 = \text{False}) \vee (x \$ 0 = x \$ 1 = \text{False})$
 $\langle \text{proof} \rangle$

lemma *tt-2:* **assumes** $dx: \text{dim-vec } x = 2$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{True}$
shows $\text{ceros-of-boolean-input } x = \{\}$
 $\langle \text{proof} \rangle$

lemma *tf-2:* **assumes** $dx: \text{dim-vec } x = 2$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{False}$
shows $\text{ceros-of-boolean-input } x = \{1\}$
 $\langle \text{proof} \rangle$

lemma *ft-2:* **assumes** $dx: \text{dim-vec } x = 2$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{True}$
shows $\text{ceros-of-boolean-input } x = \{0\}$
 $\langle \text{proof} \rangle$

lemma *ff-2:* **assumes** $dx: \text{dim-vec } x = 2$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{False}$
shows $\text{ceros-of-boolean-input } x = \{0,1\}$
 $\langle \text{proof} \rangle$

lemma
assumes $dx: \text{dim-vec } x = 2$
shows $\text{ceros-of-boolean-input } x \in \{\{\}, \{0\}, \{1\}, \{0,1\}\}$
 $\langle \text{proof} \rangle$

Vectors of dimension 3.

lemma *less-3-cases:*
assumes $n: n < 3$ **shows** $n = 0 \vee n = 1 \vee n = (2::\text{nat})$

<proof>

lemma

dim-vec-3-cases:

assumes dx : $\dim\text{-vec } x = 3$

shows $(x \$ 0 = x \$ 1 = x \$ 2 = \text{False}) \vee (x \$ 0 = x \$ 1 = \text{False} \wedge x \$ 2 = \text{True})$

$\vee (x \$ 0 = x \$ 2 = \text{False} \wedge x \$ 1 = \text{True}) \vee (x \$ 0 = \text{False} \wedge x \$ 1 = x \$ 2 = \text{True})$

$\vee (x \$ 0 = \text{True} \wedge x \$ 1 = x \$ 2 = \text{False}) \vee (x \$ 0 = x \$ 2 = \text{True} \wedge x \$ 1 = \text{False})$

$\vee (x \$ 0 = x \$ 1 = \text{True} \wedge x \$ 2 = \text{False}) \vee (x \$ 0 = x \$ 1 = x \$ 2 = \text{True})$

<proof>

lemma *fff-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False}$

shows $\text{ceros-of-boolean-input } x = \{0,1,2\}$

<proof>

lemma *fft-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True}$

shows $\text{ceros-of-boolean-input } x = \{0,1\}$

<proof>

lemma *ftf-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{False}$

shows $\text{ceros-of-boolean-input } x = \{0,2\}$

<proof>

lemma *ftt-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{True}$

shows $\text{ceros-of-boolean-input } x = \{0\}$

<proof>

lemma *tff-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False}$

shows $\text{ceros-of-boolean-input } x = \{1,2\}$

<proof>

lemma *tft-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True}$

shows $\text{ceros-of-boolean-input } x = \{1\}$

<proof>

lemma *ttf-3*: **assumes** dx : $\dim\text{-vec } x = 3$

and be : $x \$ 0 = \text{True} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{False}$

shows $\text{ceros-of-boolean-input } x = \{2\}$

<proof>

lemma ttt-3: **assumes** $dx: \text{dim-vec } x = 3$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{True}$
shows $\text{ceros-of-boolean-input } x = \{\}$
 $\langle \text{proof} \rangle$

lemma
assumes $dx: \text{dim-vec } x = 3$
shows $\text{ceros-of-boolean-input } x \in \{\{\}, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$
 $\langle \text{proof} \rangle$

Vectors of dimension 4.

lemma less-4-cases:
assumes $n: n < 4$
shows $n = 0 \vee n = 1 \vee n = 2 \vee n = (3::\text{nat})$
 $\langle \text{proof} \rangle$

lemma
 dim-vec-4-cases:
assumes $dx: \text{dim-vec } x = 4$
shows $(x \$ 0 = x \$ 1 = x \$ 2 = x \$ 3 = \text{False}) \vee (x \$ 0 = x \$ 1 = x \$ 2 = \text{False} \wedge x \$ 3 = \text{True})$
 $\vee (x \$ 0 = x \$ 1 = x \$ 3 = \text{False} \wedge x \$ 2 = \text{True}) \vee (x \$ 0 = x \$ 1 = \text{False} \wedge x \$ 2 = x \$ 3 = \text{True})$
 $\vee (x \$ 0 = x \$ 2 = x \$ 3 = \text{False} \wedge x \$ 1 = \text{True}) \vee (x \$ 0 = x \$ 2 = \text{False} \wedge x \$ 1 = x \$ 3 = \text{True})$
 $\vee (x \$ 0 = x \$ 3 = \text{False} \wedge x \$ 1 = x \$ 2 = \text{True}) \vee (x \$ 0 = \text{False} \wedge x \$ 1 = x \$ 2 = x \$ 3 = \text{True})$
 $\vee (x \$ 0 = \text{True} \wedge x \$ 1 = x \$ 2 = x \$ 3 = \text{False}) \vee (x \$ 0 = x \$ 3 = \text{True} \wedge x \$ 1 = x \$ 2 = \text{False})$
 $\vee (x \$ 0 = x \$ 2 = \text{True} \wedge x \$ 1 = x \$ 3 = \text{False}) \vee (x \$ 0 = x \$ 2 = x \$ 3 = \text{True} \wedge x \$ 1 = \text{False})$
 $\vee (x \$ 0 = x \$ 1 = \text{True} \wedge x \$ 2 = x \$ 3 = \text{False}) \vee (x \$ 0 = x \$ 1 = x \$ 3 = \text{True} \wedge x \$ 2 = \text{False})$
 $\vee (x \$ 0 = x \$ 1 = x \$ 2 = \text{True} \wedge x \$ 3 = \text{False}) \vee (x \$ 0 = x \$ 1 = x \$ 2 = x \$ 3 = \text{True})$
 $\langle \text{proof} \rangle$

lemma fff-4: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{False}$
shows $\text{ceros-of-boolean-input } x = \{0,1,2,3\}$
 $\langle \text{proof} \rangle$

lemma fff-4: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{True}$
shows $\text{ceros-of-boolean-input } x = \{0,1,2\}$
 $\langle \text{proof} \rangle$

lemma fff-4: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{False}$

shows *ceros-of-boolean-input* $x = \{0,1,3\}$
<proof>

lemma *fftt-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{True}$
shows *ceros-of-boolean-input* $x = \{0,1\}$
<proof>

lemma *ftff-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{False}$
shows *ceros-of-boolean-input* $x = \{0,2,3\}$
<proof>

lemma *ftft-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{True}$
shows *ceros-of-boolean-input* $x = \{0,2\}$
<proof>

lemma *fttf-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{False}$
shows *ceros-of-boolean-input* $x = \{0,3\}$
<proof>

lemma *fttt-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{False} \wedge x \$ 1 = \text{True} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{True}$
shows *ceros-of-boolean-input* $x = \{0\}$
<proof>

lemma *tfff-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{False}$
shows *ceros-of-boolean-input* $x = \{1,2,3\}$
<proof>

lemma *tfft-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{False} \wedge x \$ 3 = \text{True}$
shows *ceros-of-boolean-input* $x = \{1,2\}$
<proof>

lemma *tftf-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{False}$
shows *ceros-of-boolean-input* $x = \{1,3\}$
<proof>

lemma *tftt-4*: **assumes** $dx: \text{dim-vec } x = 4$
and $be: x \$ 0 = \text{True} \wedge x \$ 1 = \text{False} \wedge x \$ 2 = \text{True} \wedge x \$ 3 = \text{True}$
shows *ceros-of-boolean-input* $x = \{1\}$
<proof>

lemma *ttff-4*: **assumes** $dx: \text{dim-vec } x = 4$

and $be: x \$ 0 = True \wedge x \$ 1 = True \wedge x \$ 2 = False \wedge x \$ 3 = False$
shows *ceros-of-boolean-input* $x = \{2,3\}$
<proof>

lemma *tftf-4*: **assumes** $dx: dim-vec\ x = 4$
and $be: x \$ 0 = True \wedge x \$ 1 = True \wedge x \$ 2 = False \wedge x \$ 3 = True$
shows *ceros-of-boolean-input* $x = \{2\}$
<proof>

lemma *tftf-4*: **assumes** $dx: dim-vec\ x = 4$
and $be: x \$ 0 = True \wedge x \$ 1 = True \wedge x \$ 2 = True \wedge x \$ 3 = False$
shows *ceros-of-boolean-input* $x = \{3\}$
<proof>

lemma *tttt-4*: **assumes** $dx: dim-vec\ x = 4$
and $be: x \$ 0 = True \wedge x \$ 1 = True \wedge x \$ 2 = True \wedge x \$ 3 = True$
shows *ceros-of-boolean-input* $x = \{\}$
<proof>

lemma
ceros-of-boolean-input-set:
assumes $dx: dim-vec\ x = 4$
shows *ceros-of-boolean-input* $x \in \{\{\}, \{0\}, \{1\}, \{2\}, \{3\}, \{0,1\}, \{0,2\}, \{0,3\}, \{1,2\}, \{1,3\}, \{2,3\},$
 $\{0,1,2\}, \{0,1,3\}, \{0,2,3\}, \{1,2,3\}, \{0,1,2,3\}\}$
<proof>

context *simplicial-complex*
begin

The simplicial complex induced by the monotone Boolean function *bool-fun-threshold-2-3* has the following explicit expression.

lemma
simplicial-complex-induced-by-monotone-boolean-function-4-bool-fun-threshold-2-3:
shows $\{\{\}, \{0\}, \{1\}, \{2\}, \{3\}, \{0,1\}, \{0,2\}, \{0,3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$
 $= simplicial-complex-induced-by-monotone-boolean-function\ 4\ bool-fun-threshold-2-3$
(is $\{\{\}, ?a, ?b, ?c, ?d, ?e, ?f, ?g, ?h, ?i, ?j\} = -)$
<proof>

end

end

theory *Bij-betw-simplicial-complex-bool-func*
imports
Simplicial-complex
begin

7 Bijection between simplicial complexes and monotone Boolean functions

context *simplicial-complex*
begin

lemma *ceros-of-boolean-input-in-set*:

assumes *s*: $\sigma \in \text{simplices}$

shows *ceros-of-boolean-input* ($\text{vec } n \ (\lambda i. i \notin \sigma)$) = σ

<proof>

lemma

assumes *sigma*: $\sigma \in \text{simplices}$

and *nempty*: $\sigma \neq \{\}$

shows *Max* $\sigma < n$

<proof>

definition *bool-vec-from-simplice* :: $\text{nat set} \Rightarrow (\text{bool vec})$

where *bool-vec-from-simplice* $\sigma = \text{vec } n \ (\lambda i. i \notin \sigma)$

lemma [*simp*]:

assumes $\sigma \in \text{simplices}$

shows *ceros-of-boolean-input* (*bool-vec-from-simplice* σ) = σ

<proof>

lemma [*simp*]:

assumes *n*: $\text{dim-vec } f = n$

shows *bool-vec-from-simplice* (*ceros-of-boolean-input* f) = f

<proof>

lemma *bool-vec-from-simplice* $\{0\} = \text{vec } n \ (\lambda i. i \notin \{0\})$

<proof>

lemma *bool-vec-from-simplice* $\{1,2\} =$

$\text{vec } n \ (\lambda i. i \notin \{1,2\})$

<proof>

lemma *simplicial-complex-implies-true*:

assumes $\sigma \in \text{simplicial-complex-induced-by-monotone-boolean-function } n \ f$

shows $f \ (\text{bool-vec-from-simplice } \sigma)$

<proof>

definition *bool-vec-set-from-simplice-set* :: $\text{nat set set} \Rightarrow (\text{bool vec}) \text{ set}$

where *bool-vec-set-from-simplice-set* $K =$

$\{\sigma. (\text{dim-vec } \sigma = n) \wedge (\exists k \in K. \sigma = \text{bool-vec-from-simplice } k)\}$

definition *boolean-function-from-simplicial-complex* :: $\text{nat set set} \Rightarrow (\text{bool vec} \Rightarrow$

bool)

where *boolean-function-from-simplicial-complex* $K =$

$(\lambda x. x \in (\text{bool-vec-set-from-simplice-set } K))$

lemma *Collect (boolean-function-from-simplicial-complex K) = (bool-vec-set-from-simplice-set K)*
<proof>

The Boolean function induced by a simplicial complex is monotone. This result is proven in Scoville as part of the proof of Proposition 6.16. The opposite direction has been proven as *boolean-functions.monotone-bool-fun n ?f \implies simplicial-complex (simplicial-complex-induced-by-monotone-boolean-function n ?f)*.

lemma
simplicial-complex-induces-monotone-bool-fun:
assumes *mon: simplicial-complex (K::nat set set)*
shows *boolean-functions.monotone-bool-fun n (boolean-function-from-simplicial-complex K)*
<proof>

lemma shows *(simplicial-complex-induced-by-monotone-boolean-function n) \in boolean-functions.monotone-bool-fun-set n \rightarrow (simplicial-complex-set::nat set set set)*
<proof>

lemma shows *boolean-function-from-simplicial-complex \in (simplicial-complex-set::nat set set set) \rightarrow boolean-functions.monotone-bool-fun-set n*
<proof>

Given a Boolean function f , if we build its associated simplicial complex and then the associated Boolean function, we obtain f .

The result holds for every Boolean function f (the premise on f being monotone can be omitted).

lemma
boolean-function-from-simplicial-complex-simplicial-complex-induced-by-monotone-boolean-function:
fixes *f :: bool vec \Rightarrow bool*
assumes *dim: v \in carrier-vec n*
shows *boolean-function-from-simplicial-complex (simplicial-complex-induced-by-monotone-boolean-function n f) v = f v*
<proof>

Given a simplicial complex K , if we build its associated Boolean function, and then the associated simplicial complex, we obtain K .

lemma
simplicial-complex-induced-by-monotone-boolean-function-boolean-function-from-simplicial-complex:
fixes *K :: nat set set*
assumes *K: simplicial-complex K*
shows *simplicial-complex-induced-by-monotone-boolean-function n*

(boolean-function-from-simplicial-complex K) = K
 ⟨proof⟩

end

end

Author: Julius Michaelis

theory *MkIfex*
imports *ROBDD.BDT*
begin

8 Converting boolean functions to BDTs

The following function builds an *'a ifex* (a binary decision tree) from a boolean function and its list of variables (note that in this development we will be using boolean functions over sets of the natural numbers of the form $\{..<n\}$).

fun *mk-ifex* :: (*'a* :: *linorder*) *boolfunc* \Rightarrow *'a list* \Rightarrow *'a ifex* **where**
mk-ifex f [] = (*if* (*const False*) *then Trueif* *else Falseif*) |
mk-ifex f (*v#vs*) = *ifex-ite*
 (*IF* *v Trueif Falseif*)
 (*mk-ifex* (*bf-restrict v True f*) *vs*)
 (*mk-ifex* (*bf-restrict v False f*) *vs*)

The result of *mk-ifex* is *ifex-ordered* and *ifex-minimal*.

lemma *mk-ifex-ro: ro-ifex (mk-ifex f vs)*
 ⟨proof⟩

To prove that *mk-ifex* has correctly captured a boolean function f , we need know that all variables that f depends on were considered by *mk-ifex*. In that regard, one troublesome aspect of *boolfunc* from *ROBDD.Bool-Func* is that it is too general: Boolean functions' assignments assign a Boolean value to any natural number, and functions are not limited to "reading" only from a finite set of variables. This for example allows for the boolean function that asks "Is the assignment true in a finite number of variables: $\lambda as. \text{finite } \{x. as\ x\}$." This function does not depend on any single variable, but on the set of all of them. A definition that proved to work despite such subtleties is that a function f only depends on the variables in set x iff for any pair of assignments that agree in x (but are arbitrary otherwise), the values of f agree:

definition *reads-inside-set f x* \equiv
 ($\forall \text{assmt1 assmt2. } (\forall p. p \in x \longrightarrow \text{assmt1 } p = \text{assmt2 } p) \longrightarrow f \text{ assmt1} = f \text{ assmt2}$)

lemma *reads-inside-set-subset: reads-inside-set f a \implies a \subseteq b \implies reads-inside-set f b*

<proof>

lemma *reads-inside-set-restrict*: $\text{reads-inside-set } f \ s \implies \text{reads-inside-set } (\text{bf-restrict } i \ v \ f)$ (*Set.remove i s*)
<proof>

lemma *collect-upd-true*: $\text{Collect } (x(y := \text{True})) = \text{insert } y \ (\text{Collect } x)$ *<proof>*

lemma *collect-upd-false*: $\text{Collect } (x(y := \text{False})) = \text{Set.remove } y \ (\text{Collect } x)$ *<proof>*

lemma *reads-none*: $\text{reads-inside-set } f \ \{\} \implies f = \text{bf-True} \vee f = \text{bf-False}$
<proof>

lemma *val-ifex-ite-subst*: $\llbracket \text{ro-ifex } i; \text{ro-ifex } t; \text{ro-ifex } e \rrbracket \implies \text{val-ifex } (\text{ifex-ite } i \ t \ e)$
 $= \text{bf-ite } (\text{val-ifex } i) \ (\text{val-ifex } t) \ (\text{val-ifex } e)$
<proof>

theorem

val-ifex-mk-ifex-equal:

$\text{reads-inside-set } f \ (\text{set } vs) \implies \text{val-ifex } (\text{mk-ifex } f \ vs) \ \text{assmt} = f \ \text{assmt}$
<proof>

end

theory *Evasive*

imports

Bij-betw-simplicial-complex-bool-func

MkIfex

begin

9 Relation between type $\text{bool } \text{vec} \Rightarrow \text{bool}$ and type $'a \ \text{boolfunc}$

definition *vec-to-boolfunc* :: $\text{nat} \Rightarrow (\text{bool } \text{vec} \Rightarrow \text{bool}) \Rightarrow (\text{nat } \text{boolfunc})$
where $\text{vec-to-boolfunc } n \ f = (\lambda i. f \ (\text{vec } n \ i))$

lemma

ris: $\text{reads-inside-set } (\lambda i. \text{bool-fun-threshold-2-3} \ (\text{vec } 4 \ i)) \ (\text{set } [0,1,2,3])$
<proof>

lemma

shows $\text{val-ifex } (\text{mk-ifex } (\text{vec-to-boolfunc } 4 \ \text{bool-fun-threshold-2-3}) \ [0,1,2,3])$
 $= \text{vec-to-boolfunc } 4 \ \text{bool-fun-threshold-2-3}$
<proof>

For any Boolean function in dimension n , its ifex representation is *ifex-ordered* and *ifex-minimal*.

lemma *mk-ifex-boolean-function*:

fixes $f :: \text{bool } \text{vec} \Rightarrow \text{bool}$

shows *ro-ifex* (*mk-ifex* (*vec-to-boolfunc* *n f*) [0..*n*])
 ⟨*proof*⟩

Any Boolean function in dimension *n* can be seen as an expression over the underlying set of variables.

lemma

reads-inside-set-boolean-function:

fixes *f* :: *bool vec => bool*

shows *reads-inside-set* (*vec-to-boolfunc* *n f*) {..*n*}

⟨*proof*⟩

Any Boolean function of a finite dimension is equal to its ifex representation by means of *mk-ifex*.

lemma

mk-ifex-equivalence:

fixes *f* :: *bool vec => bool*

shows *val-ifex* (*mk-ifex* (*vec-to-boolfunc* *n f*) [0..*n*])

= *vec-to-boolfunc* *n f*

⟨*proof*⟩

definition *bcount-true* :: *nat => (nat => bool) => nat*

where *bcount-true* *n f* = $(\sum i = 0..<n. \text{if } f \text{ } i \text{ then } 1 \text{ else } (0::nat))$

definition *boolfunc-threshold-2-3* :: *(nat => bool) => bool*

where *boolfunc-threshold-2-3* = $(\lambda v. 2 \leq \text{bcount-true } 4 \text{ } v)$

definition *proj-2* :: *(nat => bool) => bool*

where *proj-2* = $(\lambda v. v \ 2)$

definition *proj-2-n3* :: *(nat => bool) => bool*

where *proj-2-n3* = $(\lambda v. v \ 2 \wedge \neg v \ 3)$

The following definition computes the height of a '*a ifex*' expression.

fun *height* :: '*a ifex*' => *nat*

where *height* *Trueif* = 0

| *height* *Falseif* = 0

| *height* (*IF* *v va vb*) = 1 + *max* (*height va*) (*height vb*)

Both *mk-ifex* and *height* can be used in computations.

lemma *height* (*mk-ifex* (*boolfunc-threshold-2-3*) [0,1,2,3]) = 4

⟨*proof*⟩

lemma *height* (*mk-ifex* (*proj-2*) [0,1,2,3]) = 1

⟨*proof*⟩

lemma *mk-ifex* (*proj-2*) [0] = *Falseif* ⟨*proof*⟩

lemma *height* (*mk-ifex* (*proj-2*) [0]) = 0 *<proof>*

lemma *mk-ifex* (*proj-2*) [3,2,1,0] = *IF 2 Trueif Falseif*
<proof>

lemma *mk-ifex* (*proj-2*) [0,1,2,3] = *IF 2 Trueif Falseif*
<proof>

lemma *height* (*mk-ifex* (*proj-2*) [0,1,2,3]) = 1 *<proof>*

lemma *mk-ifex* (*proj-2-n3*) [0,1,2,3] = *IF 2 (IF 3 Falseif Trueif) Falseif* *<proof>*

lemma *mk-ifex* (*bf-False::nat boolfunc*) [0,1,2,3] = *Falseif* *<proof>*

lemma *height* (*mk-ifex* (*bf-False::nat boolfunc*) [0,1,2,3]) = 0 *<proof>*

lemma *mk-ifex* (*bf-True::nat boolfunc*) [0,1,2,3] = *Trueif* *<proof>*

lemma *height* (*mk-ifex* (*bf-True::nat boolfunc*) [0,1,2,3]) = 0 *<proof>*

10 Definition of *evasive* Boolean function

Now we introduce the definition of evasive Boolean function. It is based on the height of the ifex expression of the given function. The definition is inspired by the one by Scoville [3, Ex. 6.19].

definition *evasive* :: *nat => ((nat => bool) => bool) => bool*
where *evasive n f* \equiv (*height* (*mk-ifex f [0..n]*)) = *n*

corollary *evasive 4 boolfunc-threshold-2-3* *<proof>*

lemma \neg *evasive 4 proj-2* *<proof>*

lemma \neg *evasive 4 proj-2-n3* *<proof>*

lemma \neg *evasive 4 bf-True* *<proof>*

lemma \neg *evasive 4 bf-False* *<proof>*

end

theory *ListLexorder*

imports *Main*

begin

11 Detour: Lexicographic ordering for lists

Simplicial complexes are defined as sets of sets. To conveniently run computations on them, we convert those sets to lists via *sorted-list-of-set*. This requires providing an arbitrary linear order for lists. We pick a lexicographic order.

datatype 'a :: linorder linorder-list = LinorderList 'a list

definition linorder-list-unwrap L \equiv case L of LinorderList L \Rightarrow L

fun less-eq-linorder-list-pre **where**

```
less-eq-linorder-list-pre (LinorderList []) (LinorderList []) = True |
less-eq-linorder-list-pre (LinorderList []) - = True |
less-eq-linorder-list-pre - (LinorderList []) = False |
less-eq-linorder-list-pre (LinorderList (a # as)) (LinorderList (b # bs))
= (if a = b then less-eq-linorder-list-pre (LinorderList as) (LinorderList bs) else
a < b)
```

instantiation linorder-list :: (linorder) linorder

begin

definition less-linorder-list x y \equiv

(less-eq-linorder-list-pre x y \wedge \neg less-eq-linorder-list-pre y x)

definition less-eq-linorder-list x y \equiv less-eq-linorder-list-pre x y

instance

\langle proof \rangle

end

The main product of this theory file:

definition sorted-list-of-list-set L \equiv

map linorder-list-unwrap (sorted-list-of-set (LinorderList ' L))

lemma set-sorted-list-of-list-set[simp]:

finite L \implies set (sorted-list-of-list-set L) = L

\langle proof \rangle

end

theory BDD

imports

Evasive

ROBDD.Level-Collapse

ListLexorder

begin

12 Executably converting Simplicial Complexes to BDDs

We already know how to convert a simplicial complex to a boolean function, and that to a BDT. We could trivially convert boolean functions to BDDs the same way they are converted to BDTs, but the conversion to BDTs necessarily takes an amount of steps exponential in the number of variables (vertices). The following method avoids this exponential method.

This theory does not include a proof on the run-time complexity of the conversion.

The basic idea is that each vertex in a simplicial complex corresponds to one *True* line in the truth table of the inducing Boolean function. This is captured by the following definition, which is part of the correctness assumptions of the final theorem.

definition *bf-from-sc* :: *nat set set => (bool vec => bool)*
where *bf-from-sc* *K* ≡ (λ*v*. {*i*. *i* < dim-vec *v* ∧ ¬ (vec-index *v* *i*)} ∈ *K*)

lemma *bf-from-sc*:

assumes *sc*: *simplicial-complex.simplicial-complex* *n* *K*
shows *simplicial-complex-induced-by-monotone-boolean-function* *n* (*bf-from-sc* *K*)
= *K*
⟨*proof*⟩

definition *boolfunc-from-sc* :: *nat => nat set set => nat boolfunc*
where *boolfunc-from-sc* *n* *K* ≡ λ*p*. {*i*. *i* < *n* ∧ ¬ *p* *i*} ∈ *K*

The conversion proven correct in two major steps:

- Prove that we can convert the list form of simplicial complexes to boolean functions instead of the set form (*boolfunc-from-sc-list*)
- Prove that we can convert the list form of simplicial complexes to BDDs (*boolfunc-bdd-from-sc-list*)

definition *sc-threshold-2-3* ≡ {{}, {0::nat}, {1}, {2}, {3}, {0,1}, {0,2}, {0,3}, {1,2}, {1,3}, {2,3}}

Example: The truth table (as separate lemmas) for *sc-threshold-2-3*:

lemma *hlp1*: {*i*. *i* < 4 ∧ ¬ (f(0 := *a0*, 1 := *a1*, 2 := *a2*, 3 := *a3*)) *i*} =
(if *a0* then {} else {0::nat})
∪ (if *a1* then {} else {1})
∪ (if *a2* then {} else {2})
∪ (if *a3* then {} else {3})
⟨*proof*⟩

lemma *sc-threshold-2-3-ffff*:

boolfunc-from-sc 4 sc-threshold-2-3 (a (0:=False,1:=False,2:=False,3:=False)) = False
<proof>

lemma *sc-threshold-2-3-ffft:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=False,2:=False,3:=True)) = False
<proof>

lemma *sc-threshold-2-3-fftf:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=False,2:=True,3:=False)) = False
<proof>

lemma *sc-threshold-2-3-ftff:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=True,2:=False,3:=False)) = False
<proof>

lemma *sc-threshold-2-3-tfff:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=False,3:=False)) = False
<proof>

lemma *sc-threshold-2-3-fttt:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=False,2:=True,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-ftft:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=True,2:=False,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-fttf:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=True,2:=True,3:=False)) = True
<proof>

lemma *sc-threshold-2-3-fttt:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=False,1:=True,2:=True,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-tftt:*
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=False,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-tfff*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=True,3:=False)) = True
<proof>

lemma *sc-threshold-2-3-tfft*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=True,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-tfff*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=False,3:=False)) = True
<proof>

lemma *sc-threshold-2-3-tfft*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=False,3:=True)) = True
<proof>

lemma *sc-threshold-2-3-tfff*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=True,3:=False)) = True
<proof>

lemma *sc-threshold-2-3-tttt*:

boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=True,3:=True)) = True
<proof>

lemma *boolfunc-from-sc n UNIV = bf-True*

<proof>

lemma *boolfunc-from-sc n {} = bf-False*

<proof>

This may seem like an extra step, but effectively, it means: require that all the atoms outside the vertex are true, but don't care about what's in the vertex.

lemma *boolfunc-from-sc-lazy*:

simplicial-complex.simplicial-complex n K
 $\implies \text{boolfunc-from-sc } n \ K = (\lambda p. \text{Pow } \{i. i < n \wedge \neg p \ i\} \subseteq K)$
<proof>

primrec *boolfunc-from-vertex-list :: nat list \Rightarrow nat list \Rightarrow (nat \Rightarrow bool) \Rightarrow bool*

where

boolfunc-from-vertex-list n [] = bf-True |

boolfunc-from-vertex-list n (f#fs) =

bf-and (boolfunc-from-vertex-list n fs) (if f \in set n then bf-True else bf-lit f)

lemma *boolfunc-from-vertex-list-Cons*:
 $boolfunc-from-vertex-list (a \# as) lUNIV =$
 $(\lambda v. (boolfunc-from-vertex-list as lUNIV) (v(a:=True)))$
 $\langle proof \rangle$

lemma *boolfunc-from-vertex-list-Empty*:
 $boolfunc-from-vertex-list [] lUNIV = Ball (set lUNIV)$
 $\langle proof \rangle$

lemma *boolfunc-from-vertex-list*:
 $set lUNIV = \{..<n\} \implies boolfunc-from-vertex-list a lUNIV = (\lambda p. \{i. i < n \wedge \neg$
 $p\ i\} \subseteq set a)$
 $\langle proof \rangle$

primrec *boolfunc-from-sc-list* :: $nat\ list \Rightarrow nat\ list\ list \Rightarrow (nat \Rightarrow bool) \Rightarrow bool$
where
 $boolfunc-from-sc-list lUNIV [] = bf-False \mid$
 $boolfunc-from-sc-list lUNIV (l\#L) =$
 $bf-or (boolfunc-from-sc-list lUNIV L) (boolfunc-from-vertex-list l lUNIV)$

lemma *boolfunc-from-sc-un*:
 $boolfunc-from-sc n (a \cup b) = bf-or (boolfunc-from-sc n a) (boolfunc-from-sc n b)$
 $\langle proof \rangle$

lemma *bf-ite-const[simp]*: $bf-ite\ bf-True\ a\ b = a\ bf-ite\ bf-False\ a\ b = b$
 $\langle proof \rangle$

lemma *Pow-subset-Pow*: $Pow\ a \subseteq Pow\ b = (a \subseteq b)$
 $\langle proof \rangle$

lemma *boolfunc-from-sc-list-concat*:
 $boolfunc-from-sc-list lUNIV (a @ b) =$
 $bf-or (boolfunc-from-sc-list lUNIV a) (boolfunc-from-sc-list lUNIV b)$
 $\langle proof \rangle$

lemma *boolfunc-from-sc-list-existing-useless*:
 $a \in set\ as \implies boolfunc-from-sc-list l (a \# as) = boolfunc-from-sc-list l as$
 $\langle proof \rangle$

primrec *remove* :: $'a \Rightarrow 'a\ list \Rightarrow 'a\ list$
where
 $remove\ a\ [] = [] \mid$
 $remove\ a\ (a1 \# as) = (if\ a = a1\ then\ []\ else\ [a1]) @ remove\ a\ as$

lemma *set-remove[simp]*: $set (remove\ a\ as) = set\ as - \{a\}$
 $\langle proof \rangle$

lemma *remove-concat[simp]*: $remove\ a\ (a1 @ a2) = remove\ a\ a1 @ remove\ a\ a2$

<proof>

lemma *boolfunc-from-sc-list-dedup1:*

boolfunc-from-sc-list l (a # as) = boolfunc-from-sc-list l (a # remove a as)

<proof>

lemma *boolfunc-from-sc-list-reorder:*

set a = set b \implies boolfunc-from-sc-list l a = boolfunc-from-sc-list l b

<proof>

lemma *boolfunc-from-sc-list:*

*set LUNIV = {..*n*::nat}*

*\implies simplicial-complex.simplicial-complex *n* (set ‘ set *L*)*

*\implies boolfunc-from-sc-list LUNIV *L* = boolfunc-from-sc *n* (set ‘ set *L*)*

<proof>

lemma *boolfunc-from-sc-alt: boolfunc-from-sc *n* K = vec-to-boolfunc *n* (bf-from-sc K)*

<proof>

primrec *bdd-from-vertex-list :: nat list \implies nat list \implies bddi \implies (nat \times bddi) Heap*

where

*bdd-from-vertex-list *n* [] s = tci s |*

*bdd-from-vertex-list *n* (f#fs) s = do {*

*(f, s) \leftarrow if f \in set *n* then tci s else litci f s;*

*(fs, s) \leftarrow bdd-from-vertex-list *n* fs s;*

andci fs f s

}

primrec *bdd-from-sc-list :: nat list \implies nat list list \implies bddi \implies (nat \times bddi) Heap*

where

bdd-from-sc-list LUNIV [] s = fci s |

bdd-from-sc-list LUNIV (l#L) s = do {

(l, s) \leftarrow bdd-from-vertex-list l LUNIV s;

(L, s) \leftarrow bdd-from-sc-list LUNIV L s;

orci L l s

}

definition *nat-list-from-vertex v \equiv sorted-list-of-set v*

definition *nat-list-from-sc K \equiv sorted-list-of-list-set (nat-list-from-vertex ‘ K)*

definition *ex-2-3 \equiv do {*

s \leftarrow emptyci;

(ex, s) \leftarrow bdd-from-sc-list [0, 1, 2, 3] (nat-list-from-sc sc-threshold-2-3) s;

graphifyci "threshold-two-three" ex s

}

lemma *nat-list-from-vertex*:

assumes *finite l*

shows $set (nat-list-from-vertex l) = \{i . i \in l\}$

<proof>

lemma

finite-sorted-list-of-set:

assumes *finite L*

shows *finite (sorted-list-of-set ' L)*

<proof>

lemma *nat-list-from-sc*:

assumes *L: finite L*

and *l: $\forall l \in L. finite l$*

shows $set ' set (nat-list-from-sc (L :: nat set set)) = \{\{i . i \in l\} \mid l. l \in L\}$

<proof>

definition *ex-false* $\equiv do \{$

s $\leftarrow emptyci;$

$(ex, s) \leftarrow bdd-from-sc-list [0, 1, 2, 3] (nat-list-from-sc \{\}) s;$

graphifyci "false" ex s

$\}$

definition *ex-true* $\equiv do \{$

s $\leftarrow emptyci;$

$(ex, s) \leftarrow bdd-from-sc-list [0, 1, 2, 3]$

$(nat-list-from-sc$

$\{\{\}, \{0\}, \{1\}, \{2\}, \{3\},$

$\{0, 1\}, \{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\},$

$\{0, 1, 2\}, \{0, 1, 3\}, \{0, 2, 3\}, \{1, 2, 3\}, \{0, 1, 2, 3\}\}) s;$

graphifyci "true" ex s

$\}$

definition *another-ex* $\equiv do \{$

s $\leftarrow emptyci;$

$(ex, s) \leftarrow bdd-from-sc-list [0, 1, 2, 3]$

$(nat-list-from-sc$

$\{\{\}, \{0\}, \{1\}, \{2\}, \{3\},$

$\{0, 1\}, \{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\},$

$\{0, 1, 2\}, \{0, 1, 3\}, \{0, 2, 3\}, \{1, 2, 3\}\}) s;$

graphifyci "another-ex" ex s

$\}$

definition *one-another-ex* $\equiv do \{$

s $\leftarrow emptyci;$

$(ex, s) \leftarrow bdd-from-sc-list [0, 1, 2, 3]$

```

      (nat-list-from-sc
       {{},{0},{1},{2},{3},
        {0,1},{0,2},{0,3},{1,2},{1,3},{2,3},
        {0,1,2},{0,1,3},{1,2,3}}) s;
    graphifyci "one-another-ex" ex s
  }

```

lemma *bf-ite-direct[simp]*: *bf-ite i bf-True bf-False = i* *<proof>*

lemma *andciI*: *node-relator (tb, tc) rp \implies node-relator (eb, ec) rp \implies rq \subseteq rp*
 \implies
<bdd-relator rp s> andci tc ec s < $\lambda(r,s').$ bdd-relator (insert (bf-and tb eb,r)
rq) s'>
<proof>

lemma *bdd-from-vertex-list[sep-heap-rules]*:
shows *<bdd-relator rp s>*
bdd-from-vertex-list n l s
< $\lambda(r,s').$ bdd-relator (insert (boolfunc-from-vertex-list n l, r) rp) s'>
<proof>

lemma *boolfunc-bdd-from-sc-list*:
shows *<bdd-relator rp s>*
bdd-from-sc-list UNIV K s
< $\lambda(r,s').$ bdd-relator (insert (boolfunc-from-sc-list UNIV K, r) rp) s'>
<proof>

lemma *map-map-idI*: *($\bigwedge x. x \in \bigcup(\text{set } ' \text{set } l) \implies f x = x \implies \text{map } (\text{map } f) l = l$)*
<proof>

definition

bdd-from-sc K n \equiv bdd-from-sc-list (nat-list-from-vertex {.. n }) (nat-list-from-sc K)

theorem *bdd-from-sc*:

assumes *simplicial-complex.simplicial-complex n (K :: nat set set)*
shows *<bdd-relator rp s>*
bdd-from-sc K n s
< $\lambda(r,s').$ bdd-relator (insert (vec-to-boolfunc n (bf-from-sc K), r) rp) s'>
<proof>

code-identifier

```

code-module Product-Type  $\rightarrow$  (SML) IBDD
  and (OCaml) IBDD and (Haskell) IBDD
| code-module Typerep  $\rightarrow$  (SML) IBDD
  and (OCaml) IBDD and (Haskell) IBDD
| code-module String  $\rightarrow$  (SML) IBDD
  and (OCaml) IBDD and (Haskell) IBDD

```

```

export-code open bdd-from-sc ex-2-3 ex-false ex-true another-ex one-another-ex
  in Haskell module-name IBDD file BDD

export-code bdd-from-sc ex-2-3
  in SML module-name IBDD file SMLBDD

end

theory Binary-operations
  imports Bij-betw-simplicial-complex-bool-func
begin

```

13 Binary operations over Boolean functions and simplicial complexes

In this theory some results on binary operations over Boolean functions and their relationship to operations over the induced simplicial complexes are presented. We follow the presentation by Chastain and Scoville [1, Sect. 1.1].

definition *bool-fun-or* :: $\text{nat} \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \Rightarrow (\text{bool vec} \Rightarrow \text{bool})$
where $(\text{bool-fun-or } n \ f \ g) \equiv (\lambda x. f \ x \ \vee \ g \ x)$

definition *bool-fun-and* :: $\text{nat} \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \Rightarrow (\text{bool vec} \Rightarrow \text{bool})$
where $(\text{bool-fun-and } n \ f \ g) \equiv (\lambda x. f \ x \ \wedge \ g \ x)$

lemma *eq-union-or*:

simplicial-complex-induced-by-monotone-boolean-function $n \ (\text{bool-fun-or } n \ f \ g)$
 $= \text{simplicial-complex-induced-by-monotone-boolean-function } n \ f$
 $\cup \text{simplicial-complex-induced-by-monotone-boolean-function } n \ g$
(is $?sc \ n \ (?bf\text{-or } n \ f \ g) = ?sc \ n \ f \cup ?sc \ n \ g)$
 $\langle \text{proof} \rangle$

lemma *eq-inter-and*:

simplicial-complex-induced-by-monotone-boolean-function $n \ (\text{bool-fun-and } n \ f \ g)$
 $= \text{simplicial-complex-induced-by-monotone-boolean-function } n \ f$
 $\cap \text{simplicial-complex-induced-by-monotone-boolean-function } n \ g$
(is $?sc \ n \ (?bf\text{-and } n \ f \ g) = ?sc \ n \ f \cap ?sc \ n \ g)$
 $\langle \text{proof} \rangle$

definition *bool-fun-ast* :: $(\text{nat} \times \text{nat}) \Rightarrow (\text{bool vec} \Rightarrow \text{bool}) \times (\text{bool vec} \Rightarrow \text{bool})$
 $\Rightarrow (\text{bool vec} \times \text{bool vec} \Rightarrow \text{bool})$
where $(\text{bool-fun-ast } n \ f) \equiv (\lambda (x,y). (fst \ f \ x) \ \wedge \ (snd \ f \ y))$

definition

simplicial-complex-induced-by-monotone-boolean-function-ast
 $:: (nat \times nat) \Rightarrow ((bool\ vec \times bool\ vec \Rightarrow bool)) \Rightarrow (nat\ set * nat\ set)\ set$
where *simplicial-complex-induced-by-monotone-boolean-function-ast* $n\ f =$
 $\{z. \exists x\ y. dim\text{-}vec\ x = fst\ n \wedge dim\text{-}vec\ y = snd\ n \wedge f\ (x, y)$
 $\wedge ((ceros\text{-}of\text{-}boolean\text{-}input\ x), (ceros\text{-}of\text{-}boolean\text{-}input\ y)) = z\}$

lemma *fst-es-simplice*:

$a \in simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\text{-}ast\ n\ f$
 $\implies (\exists x\ y. f\ (x, y) \wedge (ceros\text{-}of\text{-}boolean\text{-}input\ x) = fst(a))$
 $\langle proof \rangle$

lemma *snd-es-simplice*:

$a \in simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\text{-}ast\ n\ f$
 $\implies (\exists x\ y. f\ (x, y) \wedge (ceros\text{-}of\text{-}boolean\text{-}input\ y) = snd(a))$
 $\langle proof \rangle$

definition *set-ast* $:: (nat\ set)\ set \Rightarrow (nat\ set)\ set \Rightarrow ((nat\ set * nat\ set)\ set)$

where *set-ast* $A\ B \equiv \{c. \exists a \in A. \exists b \in B. c = (a, b)\}$

definition *set-fst* $:: (nat * nat)\ set \Rightarrow nat\ set$

where *set-fst* $AB = \{a. \exists ab \in AB. a = fst\ ab\}$

lemma *set-fst-simp* [*simp*]:

assumes $y \neq \{\}$
shows $set\text{-}fst\ (x \times y) = x$
 $\langle proof \rangle$

definition *set-snd* $:: (nat * nat)\ set \Rightarrow nat\ set$

where *set-snd* $AB = \{b. \exists ab \in AB. b = snd(ab)\}$

lemma

simplicial-complex-ast-implies-fst-true:

assumes $\gamma \in simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\text{-}ast\ nn$
 $(bool\text{-}fun\text{-}ast\ nn\ f)$
shows $fst\ f\ (simplicial\text{-}complex.\text{bool}\text{-}vec\text{-}from\text{-}simplice\ (fst\ nn)\ (fst\ \gamma))$
 $\langle proof \rangle$

lemma

simplicial-complex-ast-implies-snd-true:

assumes $\gamma \in simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\text{-}ast\ nn$
 $(bool\text{-}fun\text{-}ast\ nn\ f)$
shows $snd\ f\ (simplicial\text{-}complex.\text{bool}\text{-}vec\text{-}from\text{-}simplice\ (snd\ nn)\ (snd\ \gamma))$
 $\langle proof \rangle$

lemma *eq-ast*:

simplicial-complex-induced-by-monotone-boolean-function-ast $(n, m)\ (bool\text{-}fun\text{-}ast$
 $(n, m)\ f)$
 $= set\text{-}ast\ (simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\ n\ (fst\ f))$
 $(simplicial\text{-}complex\text{-}induced\text{-}by\text{-}monotone\text{-}boolean\text{-}function\ m\ (snd\ f))$

<proof>

end

References

- [1] E. J. Chastain and N. A. Scoville. Homology of Boolean functions and the complexity of simplicial homology. <https://nanopdf.com/download/homology-of-boolean-functions-and-the-complexity-of-simplicial-pdf>.
- [2] J. Michaelis, M. Haslbeck, P. Lammich, and L. Hupel. Algorithms for reduced ordered binary decision diagrams. *Archive of Formal Proofs*, Apr. 2016. <https://isa-afp.org/entries/ROBDD.html>, Formal proof development.
- [3] N. A. Scoville. *Discrete Morse Theory*, volume 90 of *Student Mathematical Library*. American Mathematical Society, 2019.