Simplicial complexes and Boolean functions*

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

March 19, 2025

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* ⇒ *bool* and type *'a boolfunc* **18**

10 Definition of *evasive* 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* ∈ *carrier-vec n* → (*UNIV*::*bool set*)}

definition *monotone-bool-fun* :: (*bool vec* => *bool*) => *bool*
where *monotone-bool-fun* ≡ (*mono-on* (*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* = *True*

definition *bool-fun-bot* :: *bool vec* => *bool*
where *bool-fun-bot f* = *False*

end

3 Threshold function

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

lemma *vec-index* (*vec* (*5*::*nat*) (λ*i*. *False*)) *2* = *False*
  ⟨*proof*⟩

lemma *vec-index* (*vec* (*5*::*nat*) (λ*i*. *True*)) *3* = *True*
  ⟨*proof*⟩

lemma *count-true* (*vec* (*1*::*nat*) (λ*i*. *True*)) = *1*
  ⟨*proof*⟩

lemma *count-true* (*vec* (*2*::*nat*) (λ*i*. *True*)) = *2*
  ⟨*proof*⟩

lemma *count-true* (*vec* (*5*::*nat*) (λ*i*. *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* = (λ*v*. *if i* ≤ *count-true v then True else False*)

context *boolean-functions*
begin

lemma *mono-on UNIV bool-fun-top*
  ⟨*proof*⟩

lemma *monotone-bool-fun bool-fun-top*
  ⟨*proof*⟩

lemma *mono-on UNIV bool-fun-bot*
  ⟨*proof*⟩

lemma *monotone-bool-fun bool-fun-bot*
  ⟨*proof*⟩

lemma
  monotone-count-true:
  assumes *ulev*: (*u*::*bool vec*) ≤ *v*
  shows *count-true u* ≤ *count-true v*
  ⟨*proof*⟩

The threshold function is monotone.

lemma
monotone-threshold:
assumes *ulev*: (*u*::*bool vec*) ≤ *v*
shows *bool-fun-threshold n u* ≤ *bool-fun-threshold n v*
⟨*proof*⟩

lemma
assumes (*u*::*bool vec*) ≤ *v*
and *n* < *dim-vec u*
shows *bool-fun-threshold n u* ≤ *bool-fun-threshold n v*
⟨*proof*⟩

lemma *mono-on UNIV* (*bool-fun-threshold n*)
⟨*proof*⟩

lemma *monotone-bool-fun* (*bool-fun-threshold n*)
⟨*proof*⟩

end

end

theory *Simplicial-complex*
imports
Boolean-functions
begin

4 Simplicial Complexes

lemma *Pow-singleton*: *Pow* {*a*} = {{},{*a*}} ⟨*proof*⟩

lemma *Pow-pair*: *Pow* {*a*,*b*} = {{},{*a*},{*b*},{*a*,*b*}} ⟨*proof*⟩

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 {} ∈ *simplices*
⟨*proof*⟩

lemma {*0..<n*} ∈ *simplices*
⟨*proof*⟩

lemma *finite-simplex*:
assumes *σ* ∈ *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* $K \equiv$ $(\forall \sigma \in K.\ (\sigma \in simplices) \wedge (Pow\ \sigma) \subseteq K)$

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::*nat set set*
assumes k: *simplicial-complex* K
shows $K = \{\} \vee \{\} \in K$ ⟨*proof*⟩

lemma
simplicial-complex-monotone:
fixes K::*nat set set*
assumes k: *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*::*nat*) < n
shows *simplicial-complex* {{},{*0*},{*1*},{*2*},{*3*}}
⟨*proof*⟩

lemma ¬ *simplicial-complex* {{*0*,*1*},{*1*}}
⟨*proof*⟩

Another example of simplicial complex with five simplices.

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

⟨*proof*⟩

Another example of simplicial complex with ten simplices.

lemma
assumes *three*: (*3*::*nat*) < *n*
shows *simplicial-complex*
{{*2*,*3*},{*1*,*3*},{*1*,*2*},{*0*,*3*},{*0*,*2*},{*3*},{*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* :: *bool vec* => *nat set*
where *ceros-of-boolean-input v* = {*x*. *x* < *dim-vec v* ∧ *vec-index v x* = *False*}

lemma
ceros-of-boolean-input-l-dim:
assumes *a*: *a* ∈ *ceros-of-boolean-input v*
shows *a* < *dim-vec v*
⟨*proof*⟩

lemma *ceros-of-boolean-input v* = {*x*. *x* < *dim-vec v* ∧ ¬ *vec-index v x*}
⟨*proof*⟩

lemma
ceros-of-boolean-input-complementary:
shows *ceros-of-boolean-input v* = {*x*. *x* < *dim-vec v*} − {*x*. *vec-index v x*}
⟨*proof*⟩

lemma *monotone-ceros-of-boolean-input*:
fixes *r* **and** *s*::*bool vec*
assumes *r-le-s*: *r* ≤ *s*
shows *ceros-of-boolean-input s* ⊆ *ceros-of-boolean-input r*
⟨*proof*⟩

We introduce here instantiations of the typ*bool* type for the type classes class*zero* and class*one* that will simplify notation at some points:

instantiation *bool* :: {*zero*,*one*}

begin

definition
zero-bool-def: *0 == False*

definition
one-bool-def: *1 == True*

instance ⟨*proof*⟩

end

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

definition
simplicial-complex-induced-by-monotone-boolean-function
:: *nat => (bool vec => bool) => nat set set*
where *simplicial-complex-induced-by-monotone-boolean-function n f =*
$\{y.\ \exists\, x.$ *dim-vec x = n* $\wedge$ *f x* $\wedge$ *ceros-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*::*bool vec => bool*)
⊆ *Pow* (({*0..n*}::*nat set*))
⟨*proof*⟩

corollary
simplicial-complex-induced-by-monotone-boolean-function n (*v*::*bool vec => bool*)
⊆ *Pow* ((*UNIV*::*nat set*)) ⟨*proof*⟩

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*)
⟨*proof*⟩

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* = (λ*v*. *if 2* ≤ *count-true v then True else False*)

lemma *set-list-four*: **shows** {*0*..<*4*} = *set* [*0*,*1*,*2*,*3*::*nat*] ⟨*proof*⟩

lemma *comp-fun-commute-lambda*:
comp-fun-commute-on UNIV ((+)
∘ (λ*i*. *if vec 4 f* $ *i then 1 else* (*0*::*nat*)))
⟨*proof*⟩

lemma *bool-fun-threshold-2-3*
(*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*)) = *True*
⟨*proof*⟩

lemma
0 ∉ *ceros-of-boolean-input* (*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*))
and *1* ∉ *ceros-of-boolean-input* (*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*))
and *2* ∈ *ceros-of-boolean-input* (*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*))
and *3* ∈ *ceros-of-boolean-input* (*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*))
and {*2*,*3*} ⊆ *ceros-of-boolean-input* (*vec 4* (λ*i*. *if i* = *0* ∨ *i* = *1 then True else False*))
⟨*proof*⟩

lemma *bool-fun-threshold-2-3* (*vec 4* (λ*i*. *if i* = *3 then True else False*)) = *False*
⟨*proof*⟩

lemma *bool-fun-threshold-2-3* (*vec 4* (λ*i*. *if i* = *0 then False else True*))
⟨*proof*⟩

6 The simplicial complex induced by the threshold function

lemma
empty-set-in-simplicial-complex-induced:
{} ∈ *simplicial-complex-induced-by-monotone-boolean-function 4 bool-fun-threshold-2-3*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

lemma

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

lemma *card-ceros-count-UNIV*:
shows *card* (*ceros-of-boolean-input a*) + *count-true* ((*a*::*bool vec*)) = *dim-vec a*
⟨*proof*⟩

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*: *dim-vec x = 2*
shows $(x\ \$\ 0 = x\ \$\ 1 = True) \vee (x\ \$\ 0 = False \wedge x\ \$\ 1 = True)$
$\vee\ (x\ \$\ 0 = True \wedge x\ \$\ 1 = False) \vee (x\ \$\ 0 = x\ \$\ 1 = False)$
⟨*proof*⟩

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

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

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

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

lemma
assumes *dx*: *dim-vec x = 2*
shows *ceros-of-boolean-input x* ∈ {{},{*0*},{*1*},{*0*,*1*}}
⟨*proof*⟩

Vectors of dimension 3.

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

⟨*proof*⟩

lemma
dim-vec-3-cases:
assumes *dx*: *dim-vec x = 3*
shows (*x $ 0 = x $ 1 = x $ 2 = False*) ∨ (*x $ 0 = x $ 1 = False ∧ x $ 2 = True*)
∨ (*x $ 0 = x $ 2 = False ∧ x $ 1 = True*) ∨ (*x $ 0 = False ∧ x $ 1 = x $ 2 = True*)
∨ (*x $ 0 = True ∧ x $ 1 = x $ 2 = False*) ∨ (*x $ 0 = x $ 2 = True ∧ x $ 1 = False*)
∨ (*x $ 0 = x $ 1 = True ∧ x $ 2 = False*) ∨ (*x $ 0 = x $ 1 = x $ 2 = True*)
⟨*proof*⟩

lemma *fff-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = False*
shows *ceros-of-boolean-input x = {0,1,2}*
⟨*proof*⟩

lemma *fft-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = True*
shows *ceros-of-boolean-input x = {0,1}*
⟨*proof*⟩

lemma *ftf-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = False*
shows *ceros-of-boolean-input x = {0,2}*
⟨*proof*⟩

lemma *ftt-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = True*
shows *ceros-of-boolean-input x = {0}*
⟨*proof*⟩

lemma *tff-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = False*
shows *ceros-of-boolean-input x = {1,2}*
⟨*proof*⟩

lemma *tft-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = True*
shows *ceros-of-boolean-input x = {1}*
⟨*proof*⟩

lemma *ttf-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = True ∧ x $ 1 = True ∧ x $ 2 = False*
shows *ceros-of-boolean-input x = {2}*
⟨*proof*⟩

lemma *ttt-3*: **assumes** *dx*: *dim-vec x = 3*
and *be*: *x $ 0 = True ∧ x $ 1 = True ∧ x $ 2 = True*
shows *ceros-of-boolean-input x = {}*
⟨*proof*⟩

lemma
assumes *dx*: *dim-vec x = 3*
shows *ceros-of-boolean-input x ∈ {{},{0},{1},{2},{0,1},{0,2},{1,2},{0,1,2}}*
⟨*proof*⟩

Vectors of dimension 4.

lemma *less-4-cases*:
assumes *n*: *n < 4*
shows *n = 0 ∨ n = 1 ∨ n = 2 ∨ n = (3::nat)*
⟨*proof*⟩

lemma
dim-vec-4-cases:
assumes *dx*: *dim-vec x = 4*
shows (*x $ 0 = x $ 1 = x $ 2 = x $ 3 = False*) ∨ (*x $ 0 = x $ 1 = x $ 2 = False ∧ x $ 3 = True*)
∨ (*x $ 0 = x $ 1 = x $ 3 = False ∧ x $ 2 = True*) ∨ (*x $ 0 = x $ 1 = False ∧ x $ 2 = x $ 3 = True*)
∨ (*x $ 0 = x $ 2 = x $ 3 = False ∧ x $ 1 = True*) ∨ (*x $ 0 = x $ 2 = False ∧ x $ 1 = x $ 3 = True*)
∨ (*x $ 0 = x $ 3 = False ∧ x $ 1 = x $ 2 = True*) ∨ (*x $ 0 = False ∧ x $ 1 = x $ 2 = x $ 3 = True*)
∨ (*x $ 0 = True ∧ x $ 1 = x $ 2 = x $ 3 = False*) ∨ (*x $ 0 = x $ 3 = True ∧ x $ 1 = x $ 2 = False*)
∨ (*x $ 0 = x $ 2 = True ∧ x $ 1 = x $ 3 = False*) ∨ (*x $ 0 = x $ 2 = x $ 3 = True ∧ x $ 1 = False*)
∨ (*x $ 0 = x $ 1 = True ∧ x $ 2 = x $ 3 = False*) ∨ (*x $ 0 = x $ 1 = x $ 3 = True ∧ x $ 2 = False*)
∨ (*x $ 0 = x $ 1 = x $ 2 = True ∧ x $ 3 = False*) ∨ (*x $ 0 = x $ 1 = x $ 2 = x $ 3 = True*)
⟨*proof*⟩

lemma *ffff-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = False ∧ x $ 3 = False*
shows *ceros-of-boolean-input x = {0,1,2,3}*
⟨*proof*⟩

lemma *ffft-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = False ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {0,1,2}*
⟨*proof*⟩

lemma *fftf-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = True ∧ x $ 3 = False*

shows *ceros-of-boolean-input x = {0,1,3}*
⟨*proof*⟩

lemma *fftt-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = False ∧ x $ 2 = True ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {0,1}*
⟨*proof*⟩

lemma *ftff-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = False ∧ x $ 3 = False*
shows *ceros-of-boolean-input x = {0,2,3}*
⟨*proof*⟩

lemma *ftft-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = False ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {0,2}*
⟨*proof*⟩

lemma *fttf-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = True ∧ x $ 3 = False*
shows *ceros-of-boolean-input x = {0,3}*
⟨*proof*⟩

lemma *fttt-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = False ∧ x $ 1 = True ∧ x $ 2 = True ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {0}*
⟨*proof*⟩

lemma *tfff-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = False ∧ x $ 3 = False*
shows *ceros-of-boolean-input x = {1,2,3}*
⟨*proof*⟩

lemma *tfft-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = False ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {1,2}*
⟨*proof*⟩

lemma *tftf-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = True ∧ x $ 3 = False*
shows *ceros-of-boolean-input x = {1,3}*
⟨*proof*⟩

lemma *tftt-4*: **assumes** *dx*: *dim-vec x = 4*
and *be*: *x $ 0 = True ∧ x $ 1 = False ∧ x $ 2 = True ∧ x $ 3 = True*
shows *ceros-of-boolean-input x = {1}*
⟨*proof*⟩

lemma *ttff-4*: **assumes** *dx*: *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 *ttft-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 *tttf-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* ∈ {{},{*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*: $\sigma \in$ *simplices*
  shows *ceros-of-boolean-input* (*vec n* ($\lambda i.\ i \notin \sigma$)) = σ
  ⟨*proof*⟩

lemma
  assumes *sigma*: $\sigma \in$ *simplices*
  and *nempty*: $\sigma \neq \{\}$
  shows *Max* $\sigma < n$
⟨*proof*⟩

definition *bool-vec-from-simplice* :: *nat set* => (*bool vec*)
  where *bool-vec-from-simplice* σ = *vec n* ($\lambda i.\ i \notin \sigma$)

lemma [*simp*]:
  assumes $\sigma \in$ *simplices*
  shows *ceros-of-boolean-input* (*bool-vec-from-simplice* σ) = σ
  ⟨*proof*⟩

lemma [*simp*]:
  assumes *n*: *dim-vec f* = *n*
  shows *bool-vec-from-simplice* (*ceros-of-boolean-input f*) = *f*
  ⟨*proof*⟩

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

lemma *bool-vec-from-simplice* {*1,2*} =
    vec n ($\lambda i.\ i \notin \{1,2\}$)
  ⟨*proof*⟩

lemma *simplicial-complex-implies-true*:
  assumes $\sigma \in$ *simplicial-complex-induced-by-monotone-boolean-function n f*
  shows *f* (*bool-vec-from-simplice* σ)
  ⟨*proof*⟩

definition *bool-vec-set-from-simplice-set* :: *nat set set* => (*bool vec*) *set*
  where *bool-vec-set-from-simplice-set K* =
    {σ. (*dim-vec* σ = *n*) $\wedge$ ($\exists\, k \in K.\ \sigma$ = *bool-vec-from-simplice k*)}

definition *boolean-function-from-simplicial-complex* :: *nat set set* => (*bool vec* => *bool*)
  where *boolean-function-from-simplicial-complex K* =

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

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* ⟹ *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*) ∈
boolean-functions.monotone-bool-fun-set n
→ (*simplicial-complex-set*::*nat set set set*)
⟨*proof*⟩

lemma shows *boolean-function-from-simplicial-complex*
∈ (*simplicial-complex-set*::*nat set set set*)
→ *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* ⇒ *bool*
assumes *dim*: *v* ∈ *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* ⇒ *'a list* ⇒ *'a ifex* **where**
mk-ifex f [] = (*if f* (*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: *λas. 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* ≡
(∀ *assmt1 assmt2*. (∀ *p*. *p* ∈ *x* ⟶ *assmt1 p* = *assmt2 p*) ⟶ *f assmt1* = *f assmt2*)

lemma *reads-inside-set-subset*: *reads-inside-set f a* ⟹ *a* ⊆ *b* ⟹ *reads-inside-set f b*

⟨*proof*⟩

lemma *reads-inside-set-restrict*: *reads-inside-set f s* ⟹ *reads-inside-set* (*bf-restrict i v f*) (*Set.remove i s*)
⟨*proof*⟩

lemma *collect-upd-true*: *Collect* (*x*(*y*:= *True*)) = *insert y* (*Collect x*) ⟨*proof*⟩
lemma *collect-upd-false*: *Collect* (*x*(*y*:= *False*)) = *Set.remove y* (*Collect x*) ⟨*proof*⟩

lemma *reads-none*: *reads-inside-set f* {} ⟹ *f* = *bf-True* ∨ *f* = *bf-False*
⟨*proof*⟩

lemma *val-ifex-ite-subst*: ⟦*ro-ifex i*; *ro-ifex t*; *ro-ifex e*⟧ ⟹ *val-ifex* (*ifex-ite i t e*) = *bf-ite* (*val-ifex i*) (*val-ifex t*) (*val-ifex e*)
⟨*proof*⟩

theorem
val-ifex-mk-ifex-equal:
reads-inside-set f (*set vs*) ⟹ *val-ifex* (*mk-ifex f vs*) *assmt* = *f assmt*
⟨*proof*⟩

end

theory *Evasive*
imports
Bij-betw-simplicial-complex-bool-func
MkIfex
begin

9 Relation between type *bool vec* ⇒ *bool* and type *'a boolfunc*

definition *vec-to-boolfunc* :: *nat* ⇒ (*bool vec* => *bool*) => (*nat boolfunc*)
where *vec-to-boolfunc n f* = (λ*i*. *f* (*vec n i*))

lemma
ris: *reads-inside-set* (λ*i*. *bool-fun-threshold-2-3* (*vec 4 i*)) (*set* [*0*,*1*,*2*,*3*])
⟨*proof*⟩

lemma
shows *val-ifex* (*mk-ifex* (*vec-to-boolfunc 4 bool-fun-threshold-2-3*) [*0*,*1*,*2*,*3*])
= *vec-to-boolfunc 4 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* :: *bool vec* => *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* = (∑ *i* = *0..<n*. *if f i then 1 else* (*0*::*nat*))

definition *boolfunc-threshold-2-3* :: (*nat* => *bool*) => *bool*
where *boolfunc-threshold-2-3* = (λ*v*. *2* ≤ *bcount-true 4 v*)

definition *proj-2* :: (*nat* => *bool*) => *bool*
where *proj-2* = (λ*v*. *v 2*)

definition *proj-2-n3* :: (*nat* => *bool*) => *bool*
where *proj-2-n3* = (λ*v*. *v 2* ∧ ¬ *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* ≡ (*height* (*mk-ifex f* [*0*..*n*])) = *n*

corollary *evasive 4 boolfunc-threshold-2-3* ⟨*proof*⟩

lemma ¬ *evasive 4 proj-2* ⟨*proof*⟩

lemma ¬ *evasive 4 proj-2-n3* ⟨*proof*⟩

lemma ¬ *evasive 4 bf-True* ⟨*proof*⟩

lemma ¬ *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 ≡ case L of LinorderList L ⇒ 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 ≡*
(less-eq-linorder-list-pre x y ∧ ¬ less-eq-linorder-list-pre y x)
definition *less-eq-linorder-list x y ≡ less-eq-linorder-list-pre x y*
instance
⟨*proof*⟩

end

The main product of this theory file:

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

lemma *set-sorted-list-of-list-set*[*simp*]:
finite L ⟹ set (sorted-list-of-list-set L) = L
⟨*proof*⟩

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 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-fftt*:
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-tfft*:
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=False,3:=True)) = True
⟨*proof*⟩

lemma *sc-threshold-2-3-tftf*:
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=False,2:=True,3:=False)) = True
⟨*proof*⟩

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

lemma *sc-threshold-2-3-ttff*:
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=False,3:=False)) = True
⟨*proof*⟩

lemma *sc-threshold-2-3-ttft*:
boolfunc-from-sc 4 sc-threshold-2-3 (a(0:=True,1:=True,2:=False,3:=True)) = True
⟨*proof*⟩

lemma *sc-threshold-2-3-tttf*:
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
⟹ boolfunc-from-sc n K = (λp. Pow {i. i < n ∧ ¬ p i} ⊆ K)
⟨*proof*⟩

primrec *boolfunc-from-vertex-list* :: *nat list ⇒ nat list ⇒ (nat ⇒ bool) ⇒ 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 ∈ set n then bf-True else bf-lit f)

lemma *boolfunc-from-vertex-list-Cons*:
boolfunc-from-vertex-list (*a* # *as*) *lUNIV* =
(λ*v*. (*boolfunc-from-vertex-list as lUNIV*) (*v*(*a*:=*True*)))
⟨*proof*⟩

lemma *boolfunc-from-vertex-list-Empty*:
boolfunc-from-vertex-list [] *lUNIV* = *Ball* (*set lUNIV*)
⟨*proof*⟩

lemma *boolfunc-from-vertex-list*:
set lUNIV = {..<*n*} ⟹ *boolfunc-from-vertex-list a lUNIV* = (λ*p*. {*i*. *i* < *n* ∧ ¬ *p i*} ⊆ *set a*)
⟨*proof*⟩

primrec *boolfunc-from-sc-list* :: *nat list* ⇒ *nat list list* ⇒ (*nat* ⇒ *bool*) ⇒ *bool*
where
boolfunc-from-sc-list lUNIV [] = *bf-False* |
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*∪*b*) = *bf-or* (*boolfunc-from-sc n a*) (*boolfunc-from-sc n b*)
⟨*proof*⟩

lemma *bf-ite-const*[*simp*]: *bf-ite bf-True a b* = *a bf-ite bf-False a b* = *b*
⟨*proof*⟩

lemma *Pow-subset-Pow*: *Pow a* ⊆ *Pow b* = (*a* ⊆ *b*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *boolfunc-from-sc-list-existing-useless*:
a ∈ *set as* ⟹ *boolfunc-from-sc-list l* (*a* # *as*) = *boolfunc-from-sc-list l as*
⟨*proof*⟩

primrec *remove* :: *'a* ⇒ *'a list* ⇒ *'a list*
where
remove a [] = [] |
remove a (*a1* # *as*) = (*if a* = *a1 then* [] *else* [*a1*]) @ *remove a as*

lemma *set-remove*[*simp*]: *set* (*remove a as*) = *set as* − {*a*}
⟨*proof*⟩

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* ⟹ *boolfunc-from-sc-list l a* = *boolfunc-from-sc-list l b*
⟨*proof*⟩

lemma *boolfunc-from-sc-list*:
set lUNIV = {..<*n*::*nat*}
⟹ *simplicial-complex.simplicial-complex n* (*set* ‘ *set L*)
⟹ *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* ⇒ *nat list* ⇒ *bddi* ⇒ (*nat* × *bddi*) *Heap*
where
bdd-from-vertex-list n [] *s* = *tci s* |
bdd-from-vertex-list n (*f*#*fs*) *s* = *do* {
(*f*, *s*) ← *if f* ∈ *set n then tci s else litci f s*;
(*fs*, *s*) ← *bdd-from-vertex-list n fs s*;
andci fs f s
}

primrec *bdd-from-sc-list* :: *nat list* ⇒ *nat list list* ⇒ *bddi* ⇒ (*nat* × *bddi*) *Heap*
where
bdd-from-sc-list lUNIV [] *s* = *fci s* |
bdd-from-sc-list lUNIV (*l*#*L*) *s* = *do* {
(*l*, *s*) ← *bdd-from-vertex-list l lUNIV s*;
(*L*, *s*) ← *bdd-from-sc-list lUNIV L s*;
orci L l s
}

definition *nat-list-from-vertex v* ≡ *sorted-list-of-set v*

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

definition *ex-2-3* ≡ *do* {
s ← *emptyci*;
(*ex*, *s*) ← *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* ∈ *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*: ∀ *l*∈*L*. *finite l*
shows *set* ‘ *set* (*nat-list-from-sc* (*L* :: *nat set set*)) = {{*i* . *i* ∈ *l*} | *l*. *l* ∈ *L*}
⟨*proof*⟩

definition *ex-false* ≡ *do* {
s ← *emptyci*;
(*ex*, *s*) ← *bdd-from-sc-list* [*0*, *1*, *2*, *3*] (*nat-list-from-sc* {}) *s*;
graphifyci ″*false*″ *ex s*
}

definition *ex-true* ≡ *do* {
s ← *emptyci*;
(*ex*, *s*) ← *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* ≡ *do* {
s ← *emptyci*;
(*ex*, *s*) ← *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* ≡ *do* {
s ← *emptyci*;
(*ex*, *s*) ← *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 ⟹ node-relator (eb, ec) rp ⟹ rq ⊆ rp
⟹
      <bdd-relator rp s> andci tc ec s <λ(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
  <λ(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 lUNIV K s
  <λ(r,s'). bdd-relator (insert (boolfunc-from-sc-list lUNIV K, r) rp) s'>
⟨proof⟩

lemma map-map-idI: (⋀x. x ∈ ⋃(set ' set l) ⟹ f x = x) ⟹ map (map f) l = l
  ⟨proof⟩

definition
  bdd-from-sc K n ≡ 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
  <λ(r,s'). bdd-relator (insert (vec-to-boolfunc n (bf-from-sc K), r) rp) s'>
⟨proof⟩

code-identifier
  code-module Product-Type ⇀ (SML) IBDD
    and (OCaml) IBDD and (Haskell) IBDD
  | code-module Typerep ⇀ (SML) IBDD
    and (OCaml) IBDD and (Haskell) IBDD
  | code-module String ⇀ (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* :: *nat* ⇒ (*bool vec* ⇒ *bool*) ⇒ (*bool vec* ⇒ *bool*) ⇒ (*bool vec* ⇒ *bool*)
where (*bool-fun-or n f g*) ≡ (λ*x*. *f x* ∨ *g x*)

definition *bool-fun-and* :: *nat* ⇒ (*bool vec* ⇒ *bool*) ⇒ (*bool vec* ⇒ *bool*) ⇒ (*bool vec* ⇒ *bool*)
where (*bool-fun-and n f g*) ≡ (λ*x*. *f x* ∧ *g x*)

lemma *eq-union-or*:
simplicial-complex-induced-by-monotone-boolean-function n (*bool-fun-or n f g*)
= *simplicial-complex-induced-by-monotone-boolean-function n f*
∪ *simplicial-complex-induced-by-monotone-boolean-function n g*
(**is** *?sc n* (*?bf-or n f g*) = *?sc n f* ∪ *?sc n g*)
⟨*proof*⟩

lemma *eq-inter-and*:
simplicial-complex-induced-by-monotone-boolean-function n (*bool-fun-and n f g*)
= *simplicial-complex-induced-by-monotone-boolean-function n f*
∩ *simplicial-complex-induced-by-monotone-boolean-function n g*
(**is** *?sc n* (*?bf-and n f g*) = *?sc n f* ∩ *?sc n g*)
⟨*proof*⟩

definition *bool-fun-ast* :: (*nat* × *nat*) ⇒ (*bool vec* ⇒ *bool*) × (*bool vec* ⇒ *bool*)
⇒ (*bool vec* × *bool vec* ⇒ *bool*)
where (*bool-fun-ast n f*) ≡ (λ (*x*,*y*). (*fst f x*) ∧ (*snd f y*))

definition

simplicial-complex-induced-by-monotone-boolean-function-ast
:: (*nat* × *nat*) ⇒ ((*bool vec* × *bool vec* ⇒ *bool*)) ⇒ (*nat set* ∗ *nat set*) *set*
where *simplicial-complex-induced-by-monotone-boolean-function-ast n f* =
{*z*. ∃ *x y*. *dim-vec x* = *fst n* ∧ *dim-vec y* = *snd n* ∧ *f* (*x*, *y*)
∧ ((*ceros-of-boolean-input x*), (*ceros-of-boolean-input y*)) = *z*}

lemma *fst-es-simplice*:
a ∈ *simplicial-complex-induced-by-monotone-boolean-function-ast n f*
⟹ (∃ *x y*. *f* (*x*, *y*) ∧ (*ceros-of-boolean-input x*) = *fst*(*a*))
⟨*proof*⟩

lemma *snd-es-simplice*:
a ∈ *simplicial-complex-induced-by-monotone-boolean-function-ast n f*
⟹ (∃ *x y*. *f* (*x*, *y*) ∧ (*ceros-of-boolean-input y*) = *snd*(*a*))
⟨*proof*⟩

definition *set-ast* :: (*nat set*) *set* ⇒ (*nat set*) *set* ⇒ ((*nat set*∗*nat set*) *set*)
where *set-ast A B* ≡ {*c*. ∃ *a*∈*A*. ∃ *b*∈*B*. *c* = (*a*,*b*)}

definition *set-fst* :: (*nat*∗*nat*) *set* ⇒ *nat set*
where *set-fst AB* = {*a*. ∃ *ab*∈*AB*. *a* = *fst ab*}

lemma *set-fst-simp* [*simp*]:
assumes *y* ≠ {}
shows *set-fst* (*x* × *y*) = *x*
⟨*proof*⟩

definition *set-snd* :: (*nat*∗*nat*) *set* ⇒ *nat set*
where *set-snd AB* = {*b*. ∃ *ab*∈*AB*. *b* = *snd*(*ab*)}

lemma
simplicial-complex-ast-implies-fst-true:
assumes γ ∈ *simplicial-complex-induced-by-monotone-boolean-function-ast nn*
(*bool-fun-ast nn f*)
shows *fst f* (*simplicial-complex.bool-vec-from-simplice* (*fst nn*) (*fst* γ))
⟨*proof*⟩

lemma
simplicial-complex-ast-implies-snd-true:
assumes γ ∈ *simplicial-complex-induced-by-monotone-boolean-function-ast nn*
(*bool-fun-ast nn f*)
shows *snd f* (*simplicial-complex.bool-vec-from-simplice* (*snd nn*) (*snd* γ))
⟨*proof*⟩

lemma *eq-ast*:
simplicial-complex-induced-by-monotone-boolean-function-ast (*n*, *m*) (*bool-fun-ast*
(*n*, *m*) *f*)
= *set-ast* (*simplicial-complex-induced-by-monotone-boolean-function n* (*fst f*))
(*simplicial-complex-induced-by-monotone-boolean-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.