GAP 4 Package
Thelma

THreshold ELements, Modeling and Applications

1.0.0

2018

Victor Bovdi

Vasyl Laver

Victor Bovdi
Email: vbovdi@gmail.com
Address: Department of Mathematical Sciences
UAEU
Al Ain, United Arab Emirates

Vasyl Laver
Email: vasyl.laver@uzhnu.edu.ua
Address: Department of Mathematical Sciences
UAEU
Al Ain, United Arab Emirates

Copyright

Contents

Chapter 1

Introduction

Thelma stands for "THreshold ELements, Modelling and Applications". This package is dedicated to realization of Boolean functions by the means of the threshold elements and multilayered perceptrons. Threshold elements were introduced in [MP43]. A certain number of articles on threshold logic appeared in 60-s and 70-s, however this field is regaining a considerable interest nowadays as well (see [HST16],[Hor94], [GVKB11], [KYSV16], e.t.c).

We mostly refer to the methods proposed in [ABGG80], [GPR83], [GMB17], and [Der65].

1.1 Overview over this manual

Chapter 2 describes the functions operating with the threshold elements. They include the basic operations, the functions that verify the realizability of a given Boolean function by a single threshold element, and some iterative methods. Chapter 3 describes the functions for neural networks, built from the threshold elements.

1.2 Installation

To get the newest version of this GAP 4 package download one of the archive files

- `thelma-x.x.tar.gz`
- `thelma-x.x.tar.bz2`
- `thelma-x.x.zip`

and unpack it using

```
gunzip thelma-x.x.tar.gz; tar xvf thelma-x.x.tar
```

or

```
bzip2 -d thelma-x.x.tar.bz2; tar xvf thelma-x.x.tar
```

or

```
unzip -x thelma-x.x.zip
```

respectively.

Do this in a directory called "pkg", preferably (but not necessarily) in the "pkg" subdirectory of your GAP 4 installation. It creates a subdirectory called "thelma".

As Thelma has no additional C libraries, there is no need in any additional installation steps.

1.3 Feedback

For bug reports, feature requests and suggestions, please write us an e-mail.

Chapter 2

Threshold Elements

2.1 Basic Operations

For a given real vector $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and a threshold $T \in \mathbb{R}$, the *threshold element* is a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ defined by the following relations:

$$f(x_1, \ldots, x_n) = 1, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq T, \quad \text{and} \quad f(x_1, \ldots, x_n) = 0 \quad \text{otherwise,}$$

in which $f(x_1, \ldots, x_n)$ is the binary output (valued 0 or 1), each variable x_i is the i-th input (valued 0 or 1), and n is the number of inputs.

The vector w is the *weight* vector, and the $x = (x_1, \ldots, x_n)$ is the *input* vector. The vector $(w_1, \ldots, w_n; T)$ is called the *structure vector* (or simply the *structure*) of the threshold element.

2.1.1 ThresholdElement

▷ `ThresholdElement(Weights, Threshold)` (function)

For the list of rational numbers `Weights` and the rational `Threshold` the function `ThresholdElement` returns a threshold element with the number of inputs equal to the length of the `Weights` list.

Example

```
gap> te:=ThresholdElement([1,2],3);
< threshold element with weight vector [ 1, 2 ] and threshold 3 >
gap> Display(te);
Weight vector = [ 1, 2 ], Threshold = 3.
Threshold Element realizes the function f :
[ 0, 0 ] || 0
[ 0, 1 ] || 0
[ 1, 0 ] || 0
[ 1, 1 ] || 1
Sum of Products:[ 3 ]
```

The function `Display` outputs the stucture of the given threshold element `ThrEl` and the Sum of Products or Product of Sums representation of the function realized by `ThrEl`. For threshold elements of $n \leq 4$ variables it also prints the truth table of the realized Boolean function.

Example

```
gap> w:=[1,2,4,-4,6,8,10,-25,6,32];;
gap> T:=60;;
gap> te:=ThresholdElement(w,T);
< threshold element with weight vector [ 1, 2, 4, -4, 6, 8, 10, -25, 6, 32
 ] and threshold 60 >
gap> Display(te);
Weight vector = [ 1, 2, 4, -4, 6, 8, 10, -25, 6, 32 ], Threshold = 60.
Threshold Element realizes the function f :
Sum of Products:[ 59, 155, 185, 187, 251, 315, 379, 411, 427, 441, 443, 507, 5\
71, 667, 697, 699, 763, 827, 891, 923, 939, 953, 955, 1019 ]
```

2.1.2 IsThresholdElement

▷ `IsThresholdElement(Obj)` (function)

For the object `Obj` the function `IsThresholdElement` returns `true` if `Obj` is a threshold element (see `ThresholdElement` (2.1.1)), and `false` otherwise.

Example

```
gap> te:=ThresholdElement([1,2],3);
< threshold element with weight vector [ 1, 2 ] and threshold 3 >
gap> IsThresholdElement(te);
true
gap> IsThresholdElement([[1,2],3]);
false
```

2.1.3 OutputOfThresholdElement

▷ `OutputOfThresholdElement(ThrEl)` (function)

Let $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ be a Boolean function. The vector

$$F = (\ f(0),\ f(1),\ \ldots,\ f(2^n-1)\ )^T,$$

where $f(i)$ for each $i \in \{0,1,\ldots,2^n-1\}$ is the value of $f(x_1,\ldots,x_n)$ of the i-th row in the truth table, is called the *truth vector*.

For the threshold element `ThrEl` the function `OutputOfThresholdElement` returns the truth vector of the Boolean function, realized by `ThrEl`.

Example

```
gap> te:=ThresholdElement([1,2],3);
< threshold element with weight vector [ 1, 2 ] and threshold 3 >
gap> f:=OutputOfThresholdElement(te);
[ 0, 0, 0, 1 ]
```

2.1.4 StructureOfThresholdElement

▷ `StructureOfThresholdElement(ThrEl)` (function)

For the threshold element `ThrEl` the function `StructureOfThresholdElement` returns a structure vector `[Weights,Threshold]` (see `ThresholdElement` (2.1.1)).

Example

```
gap> te:=ThresholdElement([1,2],3);
< threshold element with weight vector [ 1, 2 ] and threshold 3 >
gap> sv:=StructureOfThresholdElement(te);
[ [ 1, 2 ], 3 ]
```

2.1.5 RandomThresholdElement

▷ `RandomThresholdElement(NumVar, Lo, Hi)` (function)

For the integers `NumVar`, `Lo`, and `Hi`, the function `RandomThresholdElement` returns a threshold element of `NumVar` variables with a pseudo random integer weight vector and an integer threshold, where both the weights and the threshold are chosen from the interval [`Lo`, `Hi`].

Example

```
gap> te:=RandomThresholdElement(4,-10,10);
< threshold element with weight vector [ 7, -8, -6, 10 ] and threshold 2 >
```

2.1.6 Comparison of Threshold Elements

▷ `Comparison of Threshold Elements(ThrEl1, ThrEl2)` (function)

Let `ThrEl1` and `ThrEl2` be two threshold elements of the same number of variables, which realize the following Boolean functions (see `ThresholdElement` (2.1.1)) f_1 and f_2, respectively. By comparison of two threshold elements we mean the comparison of the truth vectors of f_1 and f_2 (see `OutputOfThresholdElement` (2.1.3)).

Example

```
gap> te1:=ThresholdElement([1,2],3);;
gap> List(OutputOfThresholdElement(te1),Order);
[ 0, 0, 0, 1 ]
gap> te2:=ThresholdElement([1,2],0);;
gap> List(OutputOfThresholdElement(te2),Order);
[ 1, 1, 1, 1 ]
gap> te3:=ThresholdElement([1,1],2);;
gap> List(OutputOfThresholdElement(te3),Order);
[ 0, 0, 0, 1 ]
gap> te1<te2;
true
gap> te1>te2;
false
```

```
gap> te1=te3;
true
```

2.2 Single Threshold Element Realizability

One of the most important questions is whether a Boolean function can be realized by a single threshold element (STE). A Boolean function which is realizable by a STE is called a `Threshold Function`. This section is dedicated to verification of STE-realizability.

In Thelma package the user is allowed to input the Boolean functions in the following forms:
(i) as a truth vector over $GF(2)$ (see `OutputOfThresholdElement` (2.1.3));
(ii) as a string, representing the truth vector of the given Boolean function;
(iii) as a polynomial over $GF(2)$.
In the case (iii) we also need to enter the number of variables of f (for example if $f(x,y) = x$, then $n = 2$).

2.2.1 CharacteristicVectorOfFunction

▷ `CharacteristicVectorOfFunction(`*`Func`*`)` (function)

Let $f(x_1,\ldots,x_n)$ be a Boolean function. We can switch from the {0,1}-base to {-1,1}-base using the following transformation:

$$y_i = 2x_i - 1, \quad (i = 1,2,\ldots,n)$$

$$g(y_1,\ldots,y_n) = 2f(x_1,\ldots,x_n) - 1.$$

For each $i \in \{1,2,\ldots,n\}$ the i-th column of the truth table of the function $g(y_1,\ldots,y_n)$ (in {-1,1}-base) we denote by Y_i, and the truth vector of g we denote by G.

Define the following vector:

$$b = \left(Y_1 \cdot G,\ \ldots,\ Y_n \cdot G,\ \sum_{i=0}^{2^n-1} g(i) \right) \in \mathbb{R}^{n+1},$$

where $Y_k \cdot G$ is the classical inner (scalar) product for each $k \in \{1,\ldots,n\}$.

Vector b is called the *`characteristic vector`* of the Boolean function f [Der65]. Comparing the characteristic vector of the function f with the lists of characteristic vectors of all STE-realizable functions we obtain the answer wheter f is realizable by STE or not. In Thelma package we have a database of all such vectors for STE-realizable functions of $n \leq 6$ variables obtained from [Der65]. For the Boolean function `Func` the function `CharacteristicVectorOfFunction` returns a characteristic vector. There are no limitations on the cardinality of `Func`, but the database of STE-realizable functions is given only for $n \leq 6$ variables.

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];
[ 0, 0, 0, 1 ]
gap> c:=CharacteristicVectorOfFunction(f);
[ 2, 2, 2 ]
gap> f:="0001";
"0001"
```

```
gap> c:=CharacteristicVectorOfFunction(f);
[ 2, 2, 2 ]
gap> x:=Indeterminate(GF(2),"x");
x
gap> y:=Indeterminate(GF(2),"y");
y
gap> f:=x+y;
x+y
gap> c:=CharacteristicVectorOfFunction(f);
Enter the number of variables (n>=2):
2
[ 0, 0, 0 ]
```

2.2.2 IsCharacteristicVectorOfSTE

▷ `IsCharacteristicVectorOfSTE(ChVect)` (function)

For the characteristic vector `ChVect` (see `CharacteristicVectorOfFunction` (2.2.1)) the function `IsCharacteristicVectorOfSTE` returns `true` if `ChVect` is a characteristic vector of some STE-realizable Boolean function, and `false` otherwise. Note, that this function is implemented only for characteristic vectors of length not bigger than 7.

Example

```
gap> f:=x*y;
x*y
gap> c:=CharacteristicVectorOfFunction(f);
Enter the number of variables (n>=2):
2
[ 2, 2, 2 ]
gap> IsCharacteristicVectorOfSTE(c);
true
gap> f:=x+y;
x+y
gap> c:=CharacteristicVectorOfFunction(f);
Enter the number of variables (n>=2):
2
[ 0, 0, 0 ]
gap> IsCharacteristicVectorOfSTE(c);
false
```

2.2.3 IsUnateInVariable

▷ `IsUnateInVariable(Func, Var)` (function)

A Boolean function $f(x_1,\ldots,x_n)$ is `positive unate` in x_i if for all possible values of x_j with $j \neq i$ we have

$$f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n) \geq f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n).$$

A Boolean function $f(x_1,\ldots,x_n)$ is *negative unate* in x_i if

$$f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n) \geq f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n).$$

For the Boolean function `Func` and the positive integer `Var` (which represents the number of the variable) the function `IsUnateBooleanFunction` returns `true` if `Func` is unate (either positive or negative) in this variable and `false` otherwise.

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),0*Z(2),0*Z(2),Z(2)^0,Z(2)^0,0*Z(2)];
[ 0, 0, 0, 0, 0, 1, 1, 0 ]
gap> nn:=BooleanFunctionByNeuralNetwork(f);;
gap> Display(nn);
Inner Layer:
[ [[ 1, -2, 3 ], 4], [[ 1, 2, -3 ], 3] ]
Outer Layer: disjunction
Neural Network realizes the function f :
[ 0, 0, 0 ] || 0
[ 0, 0, 1 ] || 0
[ 0, 1, 0 ] || 0
[ 0, 1, 1 ] || 0
[ 1, 0, 0 ] || 0
[ 1, 0, 1 ] || 1
[ 1, 1, 0 ] || 1
[ 1, 1, 1 ] || 0
Sum of Products:[ 5, 6 ]
gap> IsUnateInVariable(f,1);
true
gap> f:="00000110";
"00000110"
gap> IsUnateInVariable(f,2);
false
gap> f:=x*y+x*z;
x*y+x*z
gap> IsUnateInVariable(f,3);
Enter the number of variables (n>=3):
3
false
```

2.2.4 IsUnateBooleanFunction

▷ `IsUnateBooleanFunction(Func)` (function)

If a Boolean function f is either positive or negative unate in each variable then it is said to be *unate* (note that some x_i may be positive unate and some negative unate to satisfy the definition of unate function). A Boolean function f is *binate* if it is not unate (i.e., is neither positive unate nor negative unate in at least one of its variables).

All threshold functions are unate. However, the converse is not true, because there are certain unate functions, that can not be realized by STE [AQR99].

For the Boolean function `Func` the function `IsUnateBooleanFunction` returns `true` if `Func` is unate and `false` otherwise.

Example

```
gap> f:=[0*Z(2),Z(2)^0,Z(2)^0,Z(2)^0];
[ 0, 1, 1, 1 ]
gap> IsUnateBooleanFunction(f);
true
gap> f:="1001";
"1001"
gap> IsUnateBooleanFunction(f);
false
gap> f:=x+y;
x+y
gap> IsUnateBooleanFunction(f);
Enter the number of variables (n>=2):
2
false
```

2.2.5 InfluenceOfVariable

▷ `InfluenceOfVariable(Func, Var)` (function)

The influence of a variable x_i measures how many times out of the total existing cases a change on that variable produces a change on the output of the function.

For the Boolean function `Func` and the positive integer `Var` the function `InfluenceOfVariable` returns a positive integer - the weighted influence of the variable `Var` (to obtain integer values we multiply the influence of the variable by 2^n, where n is the number of variables of `Func`).

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),0*Z(2),0*Z(2),Z(2)^0,Z(2)^0,0*Z(2)];;
gap> InfluenceOfVariable(f,1);
2
gap> InfluenceOfVariable(f,3);
2
gap> f:="00000110";
"00000110"
gap> InfluenceOfVariable(f,2);
2
gap> f:=x*y+x*z;
x*y+x*z
gap> InfluenceOfVariable(f,3);
Enter the number of variables (n>=3):
3
2
```

2.2.6 SelfDualExtensionOfBooleanFunction

▷ `SelfDualExtensionOfBooleanFunction(Func)` (function)

The *self-dual extension* of a Boolean function $f^n : \mathbb{Z}_2^n \to \mathbb{Z}_2$ of n variables is a Boolean function $f^{n+1} : \mathbb{Z}_2^{n+1} \to \mathbb{Z}_2$ of $n+1$ variables defined as

$$f^{n+1}(x_1, \ldots, x_n, x_{n+1}) = f^n(x_1, \ldots, x_n) \quad \text{if} \quad x_{n+1} = 0,$$

$$f^{n+1}(x_1, \ldots, x_n, x_{n+1}) = 1 - f^n(\bar{x}_1, \ldots, \bar{x}_n) \quad \text{if} \quad x_{n+1} = 1,$$

where $\bar{x}_i = x_i \oplus 1$ is the negation of the i-th variable.

Every threshold function is unate. However, in [FSAJ06] was shown that the unatness in the self-dual space of $n+1$ variables is much stronger condition.

For the Boolean function `Func` the function `SelfDualExtensionOfBooleanFunction` returns the truth vector of the self-dual extension of `Func`.

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];;
gap> fsd:=SelfDualExtensionOfBooleanFunction(f);;
gap> List(fsd,Order);
[ 0, 0, 0, 1, 0, 1, 1, 1 ]
gap> f:="0001";;
gap> fsd:=SelfDualExtensionOfBooleanFunction(f);;
gap> List(fsd,Order);
[ 0, 0, 0, 1, 0, 1, 1, 1 ]
gap> f:=x*y;;
gap> fsd:=SelfDualExtensionOfBooleanFunction(f);;
Enter the number of variables (n>=2):
2
gap> List(fsd,Order);
[ 0, 0, 0, 1, 0, 1, 1, 1 ]
```

2.2.7 SplitBooleanFunction

▷ `SplitBooleanFunction(Func, Var, Bool)` (function)

The method of splitting a function in terms of a given variable is known as Shannon decomposition and it was formally introduced in 1938 by Shannon.

Let $f(x_1, \ldots, x_n)$ be a Boolean function. Decompose f as a disjunction of the following two Boolean functions f_a and f_b defined as:

$$f_a(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \quad \text{if} \quad x_i = 0,$$

$$f_a(x_1, \ldots, x_n) = 0, \quad \text{if} \quad x_i = 1;$$

and

$$f_b(x_1, \ldots, x_n) = 0 \quad \text{if} \quad x_i = 0,$$

$$f_b(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \quad \text{if} \quad x_i = 1.$$

If are intended to use conjunction, we can apply the same equations with 1 for undetermined outputs instead of 0.

For the Boolean function `Func`, a positive integer `Var` (the number of variable), Boolean variable `Bool` (`true` for disjunction and `false` for conjunction) the function `SplitBooleanFunction` returns a list with two entries: the truth vectors of the resulting functions.

Example
```
gap> f:=[0*Z(2),Z(2)^0,Z(2)^0,0*Z(2)];;
gap> out:=SplitBooleanFunction(f,1,false);;
gap> List(out[1],Order);
[ 0, 1, 1, 1 ]
gap> List(out[2],Order);
[ 1, 1, 1, 0 ]
gap> f:="0110";
"0110"
gap> out:=SplitBooleanFunction(f,1,true);;
gap> List(out[1],Order);
[ 0, 1, 0, 0 ]
gap> List(out[2],Order);
[ 0, 0, 1, 0 ]
gap> f:=x+y;
x+y
gap> out:=SplitBooleanFunction(f,1,true);
Enter the number of variables (n>=2):
2
[ [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ] ]
```

2.2.8 KernelOfBooleanFunction

▷ `KernelOfBooleanFunction(`*Func*`)` (function)

For a Boolean function $f(x_1,\ldots,x_n)$ we define the following two sets (see [ABGG80]):

$$f^{-1}(1) = \{\, \mathbf{x} \in \mathbb{Z}_2^n \mid f(\mathbf{x}) = 1 \,\}, \quad \text{and} \quad f^{-1}(0) = \{\, \mathbf{x} \in \mathbb{Z}_2^n \mid f(\mathbf{x}) = 0 \,\}.$$

The kernel $K(f)$ of the Boolean function f is defined as

$$K(f) = f^{-1}(1), \quad \text{if} \quad |f^{-1}(1)| \geq |f^{-1}(0)|;$$

$$K(f) = f^{-1}(0), \quad \text{otherwise},$$

where $|f^{-1}(i)|$ is the cardinality of the set $f^{-1}(i)$ with $i \in \{0,1\}$.

For the Boolean function `Func` the function `KernelOfBooleanFunction` returns a list in which the first element of the output list represents the kernel, and the second element equals either 1 or 0.

Example
```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];
[ 0, 0, 0, 1 ]
gap> k:=KernelOfBooleanFunction(f);
```

```
[ [ [ 1, 1 ] ], 1 ]
gap> f:="0111";
"0111"
gap> k:=KernelOfBooleanFunction(f);
[ [ [ 0, 0 ] ], 0 ]
gap> z:=Indeterminate(GF(2),"z");
z
gap> f:=x*y+z;
x*y+z
gap> k:=KernelOfBooleanFunction(f);
Enter the number of variables n (n>=3):
3
[ [ [ 0, 0, 1 ], [ 0, 1, 1 ], [ 1, 0, 1 ], [ 1, 1, 0 ] ], 1 ]
```

2.2.9 ReducedKernelOfBooleanFunction

▷ `ReducedKernelOfBooleanFunction(Ker)` (function)

Let $f(x_1,\dots,x_n)$ be a Boolean function with the kernel $K(f) = \{ a_1,\dots,a_m \}$, where $m \leq 2^{n-1}$. The reduced kernel $K(f)_i$ of the function f relative to the element $a_i \in K(f)$ is the following set (see [ABGG80]):

$$K(f)_i = \left\{ a_1 \oplus a_i,\ a_2 \oplus a_i,\ \dots,\ a_m \oplus a_i \right\},$$

where $\oplus$ is a component-wise addition of vectors from $K(f)$ over $GF(2)$.

The reduced kernel $T(f)$ of f is the following set:

$$T(f) = \left\{ K(f)_i \mid i = 1,2,\dots,m \right\}.$$

For the $m \times n$ matrix `Ker`, which represents the kernel of some Boolean function f, the function `ReducedKernelOfBooleanFunction` returns the reduced kernel $T(f)$ of f.

Example

```
gap> ## Continuation of Example 2.2.4
gap> rk:=ReducedKernelOfBooleanFunction(k[1]);;
gap> j:=1;;
gap> for i in rk do Print(j,".\n"); Display(i); Print("\n"); j:=j+1; od;
1.
 . . .
 . 1 .
 1 . .
 1 1 1

2.
 . 1 .
 . . .
 1 1 .
 1 . 1

3.
 1 . .
 1 1 .
```

```
 . . .
 . 1 1

4.
 1 1 1
 1 . 1
 . 1 1
 . . .
```

2.2.10 IsInverseInKernel

▷ `IsInverseInKernel(Func)` (function)

Let $f(x_1,\dots,x_n)$ be a Boolean function with the kernel $K(f)$. The function `IsInverseInKernel` returns `true` if there is a pair of additive inverse vectors in $K(f)$ (this means that f is not STE-realizable, see [GPR83]) or `false` otherwise. Note that this function also accepts the kernel of the Boolean function `Func` as an input. A vector $b \in \mathbb{Z}_2^n$ is called an additive inverse to $a \in \mathbb{Z}_2^n$ if $a \oplus b = 0$.

Example

```
gap> f:=x*y+z;
x*y+z
gap> k:=KernelOfBooleanFunction(f);;
Enter the number of variables n (n>=3):
3
gap> Display(k[1]);
 . . 1
 . 1 1
 1 . 1
 1 1 .
gap> IsInverseInKernel(f);
Enter the number of variables n (n>=3):
3
true
```

2.2.11 IsKernelContainingPrecedingVectors

▷ `IsKernelContainingPrecedingVectors(Func)` (function)

A vector $a = (\alpha_1,\dots,\alpha_n) \in \mathbb{Z}_2^n$ precedes a vector $b = (\beta_1,\dots,\beta_n) \in \mathbb{Z}_2^n$ (we denote it as $a \prec b$) if $\alpha_i \le \beta_i$ for each $i = 1,\dots,n$.

For a given vector $c \in \mathbb{Z}_2^n$ denote $M_c = \{\, a \in \mathbb{Z}_2^n \mid a \prec c \,\}$.

Let $f(x_1,\dots,x_n)$ be a Boolean function with reduced kernel $T(f) = \{K(f)_j \mid j = 1,2,\dots,m\}$. If f is implemented by a single threshold element (STE), then there exists $j \in \{1,\dots,m\}$ such that

$$\forall a \in K(f)_j \qquad \text{holds} \qquad M_a \subseteq K(f)_j.$$

The function `IsKernelContainingPrecedingVectors` returns `false` for a given function `Func` if *Func* is not realizable by a single threshold element (see [GMB17]). Note that this function also accepts the kernel of the Boolean function `Func` as an input.

Example

```
gap> f:=x*y+z;
x*y+z
gap> IsKernelContainingPrecedingVectors(f);
Enter the number of variables n (n>=3):
3
false
```

2.2.12 IsRKernelBiggerOfCombSum

▷ `IsRKernelBiggerOfCombSum(`*Func*`)` (function)

Let $f(x_1,\ldots,x_n)$ be a Boolean function with reduced kernel $T(f)$. Denote

$$k_i^* = \max\left\{\ \|a\| = \textstyle\sum_{j=1}^{m} a_j \ \mid\ a = (a_1,\ldots,a_m) \in T(f)\ \right\}, \quad (i = 1,\ldots,n)$$

and

$$k_A^* = \min\left\{\ k_i^* \ \mid\ i = 1,2,\ldots,n\ \right\}.$$

If f is implemented by a single threshold element (STE), then the following condition holds:

$$|A| \geq \sum_{i=0}^{k_A^*} \binom{k_A^*}{i},$$

where $\binom{k_A^*}{i}$ is the classical binomial coefficient and $|A|$ is the cardinality of A.

For a given Boolean function `Func` the function `IsRKernelBiggerOfCombSum` returns `false` if this function is not STE-realizable (see [GMB17]). Note that this function also accepts the reduced kernel of the Boolean function `Func` as an input.

Example

```
gap> f:=x+y;
x+y
gap> IsRKernelBiggerOfCombSum(f);
Enter the number of variables n (n>=2):
2
false
```

2.2.13 BooleanFunctionBySTE

▷ `BooleanFunctionBySTE(`*Func*`)` (function)

For a given Boolean function `Func` the function `BooleanFunctionBySTE` determines whether `Func` is realizable by a single threshold element (STE). The function returns a threshold element with integer weights and integer threshold. If `Func` is not realizable by STE, it returns an empty list []. The realization of the function `BooleanFunctionBySTE` is based on algorithms, proposed in [Gec10].

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];
[ 0, 0, 0, 1 ]
gap> te:=BooleanFunctionBySTE(f);
< threshold element with weight vector [ 1, 2 ] and threshold 3 >
gap> f:="11001000";
"11001000"
gap> te:=BooleanFunctionBySTE(f);
< threshold element with weight vector [ -1, -4, -2 ] and threshold -2 >
gap> Display(last);
Weight vector = [ -1, -4, -2 ], Threshold = -2.
Threshold Element realizes the function f :
[ 0, 0, 0 ] || 1
[ 0, 0, 1 ] || 1
[ 0, 1, 0 ] || 0
[ 0, 1, 1 ] || 0
[ 1, 0, 0 ] || 1
[ 1, 0, 1 ] || 0
[ 1, 1, 0 ] || 0
[ 1, 1, 1 ] || 0
Sum of Products:[ 0, 1, 4 ]
gap> f:=x+y;
x+y
gap> te:=BooleanFunctionBySTE(f);
Enter the number of variables n (n>=2):
2
[  ]
```

2.2.14 PDBooleanFunctionBySTE

▷ `PDBooleanFunctionBySTE(Func)` (function)

Let $f(x_1,\ldots,x_n)$ be a partially defined Boolean function. We denote by `x` the positions in truth vector, where f is undefined. Then f^{-1}`(x)` is the set of Boolean vectors of n variables on which the function is undefined. The sets $f^{-1}(0)$ and $f^{-1}(1)$ are defined in `KernelOfBooleanFunction` (2.2.8). The function f is called a *threshold function* if there is an n-dimensional real vector $w=(w_1,\ldots,w_n)$ and a real threshold T such that

$$a \in f^{-1}(1) \quad \Longrightarrow \quad a \cdot w^T \geq T,$$

$$a \in f^{-1}(0) \quad \Longrightarrow \quad a \cdot w^T < T,$$

where $a \cdot w^T$ is the classical inner (scalar) product.

For the partially defined Boolean function `Func` (presented as a string, where `x` presents the undefined values) the function `PDBooleanFunctionBySTE` returns a threshold element if `Func` can be realized by STE and empty list otherwise. The realization of the function `PDBooleanFunctionBySTE` is based on the algorithm, proposed in [GPR83].

Example

```
gap> f:="1x001x0x";
"1x001x0x"
gap> te:=PDBooleanFunctionBySTE(f);
< threshold element with weight vector [ -1, -2, -3 ] and threshold -1 >
gap> List(OutputOfThresholdElement(te),Order);
[ 1, 0, 0, 0, 1, 0, 0, 0 ]
```

2.3 Iterative Training Methods

Thelma also provides a few iterative methods for threshold element training.

2.3.1 ThresholdElementTraining

▷ `ThresholdElementTraining(ThrEl, Step, Func, Max_Iter)` (function)

This is a basic iterative method for the perceptron training [Ros58]. For the threshold element `ThrEl` (which is an arbitrary threshold element for the first iteration), the positive integer `Step` (the value on which we change parameters while training the threshold element), the Boolean function `Func` and the positive integer `Max_Iter` - the maximal number of iterations, the function `ThresholdElementTraining` returns a threshold element, realizing `Func` (if such threshold element exists).

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];
[ 0, 0, 0, 1 ]
gap> te1:=RandomThresholdElement(2,-2,2);
< threshold element with weight vector [ 0, -1 ] and threshold 0 >
gap> OutputOfThresholdElement(te1);
[ 1, 0, 1, 0 ]
gap> te2:=ThresholdElementTraining(te1,1,f,100);
< threshold element with weight vector [ 2, 1 ] and threshold 3 >
gap> OutputOfThresholdElement(te2);
[ 0, 0, 0, 1 ]
```

2.3.2 ThresholdElementBatchTraining

▷ `ThresholdElementBatchTraining(ThrEl, Step, Func, Max_Iter)` (function)

For the threshold element `ThrEl` (which is an arbitrary threshold element for the first iteration), the positive integer `Step` (the value on which we change parameters while training the threshold element), the Boolean function `Func`, and the positive integer `Max_Iter` - the maximal number of iterations, the function `ThresholdElementTraining` returns a threshold element, realizing `Func` (if such threshold element exists) via batch training.

Example

```
gap> f:=[0*Z(2),0*Z(2),0*Z(2),Z(2)^0];
[ 0, 0, 0, 1 ]
gap> te1:=RandomThresholdElement(2,-2,2);
< threshold element with weight vector [ 0, 2 ] and threshold 2 >
gap> OutputOfThresholdElement(te1);
[ 0, 1, 0, 1 ]
gap> te2:=ThresholdElementBatchTraining(te1,1,f,100);
< threshold element with weight vector [ 2, 2 ] and threshold 3 >
gap> OutputOfThresholdElement(te2);
[ 0, 0, 0, 1 ]
```

2.3.3 WinnowAlgorithm

▷ `WinnowAlgorithm(Func, Step, Max_Iter)` (function)

A Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ which can be presented in the following form:

$$f(x_1,\ldots,x_n) = x_{i_1} \vee \cdots \vee x_{i_k}, \qquad (k \le n)$$

is called a *monotone disjunction*, i.e. it is a disjunction in which no variable appears negated.

If the given Boolean function f is a monotone disjunction, the *Winnow algorithm* is more efficient than the classical Perceptron training algorithm [Lit88].

For the Boolean function `Func`, which is a monotone disjunction, `WinnowAlgorithm` returns either a threshold element realizing `Func` or [] if `Func` is not trainable by `WinnowAlgorithm`. The positive ingetger `Step` which is not equal to 1 defines the value on which we change parameters while running the algorithm and the positive integer `Max_Iter` defines the maximal number of iterations.

Example

```
gap> f:=x*y+x+y;;
gap> te:=WinnowAlgorithm(f,2,100);
Enter the number of variables (n>=2):
2
< threshold element with weight vector [ 1, 1 ] and threshold 1 >
gap> OutputOfThresholdElement(te);
[ 0, 1, 1, 1 ]
```

2.3.4 Winnow2Algorithm

▷ `Winnow2Algorithm(Func, Step, Max_Iter)` (function)

For any $X \subseteq \mathbb{Z}_2^n$ and for any δ satisfying $0 < \delta \le 1$ let $F(X,\delta)$ be the class of functions from X to $\mathbb{Z}_2^n$. Assume that $F(X,\delta)$ satisfies the following condition:

for each $f \in F(X,\delta)$ there exist $\mu_1,\ldots,\mu_n \ge 0$ such that for all $(x_1,\ldots,x_n) \in X$

$$\textstyle\sum_{i=1}^n \mu_i x_i \ge 1, \quad \text{if} \quad f(x_1,\ldots,x_n) = 1$$

and

$$\sum_{i=1}^{n} \mu_i x_i \leq 1, \quad \text{if} \quad f(x_1,\ldots,x_n) = 0.$$

In other words, the inverse images of 0 and 1 are linearly separable with a minimum separation that depends on δ. Winnow2 algorithm is designed for training this class of the Boolean functions [Lit88].

For the Boolean function Func from the class of Boolean functions which is described above, the function Winnow2Algorithm returns either a threshold element which realizes Func or [] if Func is not trainable by Winnow2Algorithm. The positive integer Step which is not equal to 1 defines the value on which we change parameters while running the algorithm.

Example

```
gap> ## Conjunction can not be trained by Winnow algorithm.
gap> f:=x*y;;
gap> te:=WinnowAlgorithm(f,2,100);
Enter the number of variables (n>=2):
2
[  ]
gap> ## But in the case of Winnow2 we can obtain the desirable result.
gap> te:=Winnow2Algorithm(f,2,100);
Enter the number of variables (n>=2):
2
< threshold element with weight vector [ 1/2, 1/2 ] and threshold 1 >
gap> OutputOfThresholdElement(te);
[ 0, 0, 0, 1 ]
```

2.3.5 STESynthesis

▷ STESynthesis(*Func*) (function)

The function STESynthesis is based on the algorithm proposed in [Der65]. In each iteration we perturb an $n+1$-dimensional weight-threshold vector in such manner that the distance between the given vector and a desired weight-threshold vector, if such vector exists, is reduced. So if the Boolean function Func is STE-realizable, then this procedure will eventually yield an acceptable weight-threshold vector. Otherwise iteration process will eventually enter a limit cycle and the execution of STE_Synthesis will be stopped.

For the Boolean function Func the function STESynthesis returns a threshold element if Func is STE-realizable or an empty list otherwise.

Example

```
gap> f:=x*y+x+y;;
gap> te:=STESynthesis(f);
Enter the number of variables (n>=2):
2
< threshold element with weight vector [ 2, 2 ] and threshold 1 >
gap> Display(last);
Weight vector = [ 2, 2 ], Threshold = 1.
Threshold Element realizes the function f :
[ 0, 0 ] || 0
[ 0, 1 ] || 1
```

```
[ 1, 0 ] || 1
[ 1, 1 ] || 1
Product of Sums:[ 0 ]
```

Chapter 3

Networks of Threshold Elements

Not all Boolean functions can be realized by a single threshold element. However, all of them can be realized by a multi-layered network of threshold elements, with a number of threshold elements on a first layer and conjunction or a disjunction on the second layer. In this chapter we will decribe some functions regarding such networks.

3.1 Basic Operations

In this section we describe some operations, similar to the ones described in Section 2.1.

3.1.1 NeuralNetwork

▷ `NeuralNetwork(InnerLayer, OuterLayer)` (function)

For the list of threshold elements `InnerLayer` and the Boolean variable `OuterLayer`, which can be either `true` (for disjunction), `false` (for conjunction), or `fail` (if there is only one layer) the function `NeuralNetwork` returns a neural network built from this inputs.

Example

```
gap> te1:=ThresholdElement([1,1],1);
< threshold element with weight vector [ 1, 1 ] and threshold 1 >
gap> te2:=ThresholdElement([-1,-2],-2);
< threshold element with weight vector [ -1, -2 ] and threshold -2 >
gap> inner:=[te1,te2];
[ < threshold element with weight vector [ 1, 1 ] and threshold 1 >,
  < threshold element with weight vector [ -1, -2 ] and threshold -2 > ]
gap> nn:=NeuralNetwork(inner,false);
< neural network with
2 threshold elements on inner layer and conjunction on outer level >
gap> Display(last);
Inner Layer:
[ [[ 1, 1 ], 1], [[ -1, -2 ], -2] ]
Outer Layer: conjunction
Neural Network realizes the function f :
[ 0, 0 ] || 0
[ 0, 1 ] || 1
[ 1, 0 ] || 1
```

```
[ 1, 1 ] || 0
Sum of Products:[ 1, 2 ]
```

3.1.2 IsNeuralNetwork

▷ `IsNeuralNetwork(Obj)` (function)

For the object `Obj` the function `IsNeuralNetwork` returns `true` if `Obj` is a neural network (see `NeuralNetwork` (3.1.1)), and `false` otherwise.

Example

```
gap> ## Consider the neural network <C>nn</C> from the previous example.
gap> IsNeuralNetwork(nn);
true
```

3.1.3 OutputOfNeuralNetwork

▷ `OutputOfNeuralNetwork(NNetwork)` (function)

For the neural network `NNetwork` the function `OutputOfNeuralNetwork` returns the truth vector of the Boolean function, realized by `NNetwork`.

Example

```
gap> OutputOfNeuralNetwork(nn);
[ 0, 1, 1, 0 ]
```

3.2 Networks of Threshold Elements

In this section we consider the networks of threshold elements.

3.2.1 BooleanFunctionByNeuralNetwork

▷ `BooleanFunctionByNeuralNetwork(Func)` (function)

For the Boolean function `Func` the function `BooleanFunctionByNeuralNetwork` returns a two-layered neural network, which realizes `Func` (see `NeuralNetwork` (3.1.1)). The realization of this function is based on the algorithm proposed in [GPR83].

Example

```
gap> f:=x*y+z;
x*y+z
gap> nn:=BooleanFunctionByNeuralNetwork(f);
Enter the number of variables n (n>=3):
3
< neural network with
```

```
2 threshold elements on inner layer and disjunction on outer level >
gap> Display(last);
Inner Layer:
[ [[ -1, -2, 4 ], 2], [[ 1, 2, -3 ], 3] ]
Outer Layer: disjunction
Neural Network realizes the function f :
[ 0, 0, 0 ] || 0
[ 0, 0, 1 ] || 1
[ 0, 1, 0 ] || 0
[ 0, 1, 1 ] || 1
[ 1, 0, 0 ] || 0
[ 1, 0, 1 ] || 1
[ 1, 1, 0 ] || 1
[ 1, 1, 1 ] || 0
Sum of Products:[ 1, 3, 5, 6 ]
```

3.2.2 BooleanFunctionByNeuralNetworkDASG

▷ `BooleanFunctionByNeuralNetworkDASG(`*Func*`)` (function)

For the Boolean function `Func` the function `BooleanFunctionByNeuralNetworkDASG` returns a two-layered neural network which realizes `Func` (see `NeuralNetwork` (3.1.1)). The realization of this function is based on decomposition of `Func` by the non-unate variables with the biggest influence. The DASG algorithm (DASG - Decomposition Algorithm for Synthesis and Generalization) was proposed in [SJF08], however we use a slightly modified version of this algorithm.

Example

```
gap> f:="00000110";
"00000110"
gap> nn:=BooleanFunctionByNeuralNetworkDASG(f);;
< neural network with
  2 threshold elements on inner layer and conjunction on outer level >
gap> Display(last);
Inner Layer:
[ [[ 1, 4, 2 ], 3], [[ 1, -4, -2 ], -3] ]
Outer Layer: conjunction
Neural Network realizes the function f :
[ 0, 0, 0 ] || 0
[ 0, 0, 1 ] || 0
[ 0, 1, 0 ] || 0
[ 0, 1, 1 ] || 0
[ 1, 0, 0 ] || 0
[ 1, 0, 1 ] || 1
[ 1, 1, 0 ] || 1
[ 1, 1, 1 ] || 0
Sum of Products:[ 5, 6 ]
```

References

[ABGG80] N. Aizenberg, A. Bovdi, E. Gergo, and F. Geche. Algebraic aspects of threshold logic. *Cybernetics*, 16:188–193, 03 1980. 4, 14, 15

[AQR99] M. J. Avedillo, J. M. Quintana, and A. Rueda. *Threshold Logic*, pages 178–190. American Cancer Society, 1999. 11

[Der65] M.L. Dertouzos. *Threshold Logic: A Synthesis Approach*. M.I.T. Press research monographs. M.I.T. Press, 1965. 4, 9, 21

[FSAJ06] L. Franco, J. L. Subirats, M. Anthony, and J. M. Jerez. A new constructive approach for creating all linearly separable (threshold) functions. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 4791–4796, 2006. 13

[Gec10] F. Geche. *Analysis of Discrete Functions and Logical Schemes Synthesis in Neurobasis (in Ukrainian)*. Uzhhorod National University, 2010. 17

[GMB17] F. Geche, O. Mulesa, and V. Buchok. Verification of realizability of boolean functions by a neural element with a threshold activation function. *Eastern European Journal of Enterprise Technologies*, 1:30–40, 01 2017. 4, 17

[GPR83] F. E. Geche, V. P. Polivko, and V. I. Robotishin. Realization of Boolean functions using threshold elements. *Kibernetika (Kiev)*, (6):62–67, 1983. 4, 16, 18, 24

[GVKB11] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski. Identification of threshold functions and synthesis of threshold networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):665–677, May 2011. 4

[Hor94] E. K. Horváth. Invariance groups of threshold functions. *Acta Cybernetica*, 11(4):325–332, 1994. 4

[HST16] E. Horváth, B. Seselja, and A. Tepavcevic. A note on lattice variant of thresholdness of boolean functions. *Miskolc Mathematical Notes*, 17:293–304, 01 2016. 4

[KYSV16] N. Kulkarni, J. Yang, J. S. Seo, and S. Vrudhula. Reducing power, leakage, and area of standard-cell asics using threshold logic flip-flops. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(9):2873–2886, September 2016. 4

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, April 1988. 20, 21

[MP43] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943. 4

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958. 19

[SJF08] J. L. Subirats, J. M. Jerez, and L. Franco. A new decomposition algorithm for threshold synthesis and generalization of boolean functions. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(10):3188–3196, Nov 2008. 25

Index