Teatime capsid:

The search space of the logical function synthesis problem — application for biological systems

Athénaïs Vaginay

2020 mai 26th

How is it link to what I explain you last time?

boolean networks = network of boolean automata

- automata
- automata configuration
- system configuration
- influence
- local update functions
- updating scheme
- transition graph
- reachability

Automates
© Musée des arts et métiers-Cnam
Sylvain Pelly

How is it link to what I explain you last time?

boolean networks = network of boolean automata

- automata
- automata configuration
- system configuration
- influence
- **local update functions**
- updating scheme
- transition graph
- reachability

Automates
© Musée des arts et métiers-Cnam
Sylvain Pelly

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B} \ \forall i \in V)$$

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B} \ \forall i \in V)$$

- "a activates x" $\to x_{t+1} = f(a_t)$
  the status of a at time t is the only important input that determine the status of x at time $t + 1$

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B}\ \forall i \in V)$$

- "a activates x" $\rightarrow x_{t+1} = f(a_t) = g(a_t, b_t)$
  the status of a at time t is the only important input that determine the status of x at time $t + 1$

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B} \ \forall i \in V)$$

- "a activates x" $\rightarrow x_{t+1} = f(a_t) = g(a_t, b_t)$
  the status of a at time t is the only important input that determine the status of x at time $t+1$
- "both a and b can activate x" $\rightarrow x_{t+1} = f(a_t, b_t)$
  both the status of a and b at time t are required to determine the status of x at $t+1$

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B}\ \forall i \in V)$$

logical function synthesis, using constraints about

- the definition domain of the function (hard constraint)

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B}\ \forall i \in V)$$

logical function synthesis, using constraints about

- the definition domain of the function (hard constraint)
  "x potentially explained by a
  x potentially explained by b"
  $\rightarrow \quad x_{t+1} = f(a_t); \quad x_{t+1} = g(b_t); \quad x_{t+1} = h(a_t, b_t)$

On network of n boolean automata — formaly

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B} \ \forall i \in V)$$

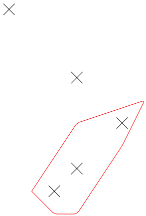
logical function synthesis, using constraints about

- the definition domain of the function (hard constraint)
- $+$ specification abouts their behavior (soft constraint to be optimized).
  TODO: explain why soft (orally?)

Goal:
Defining the search space

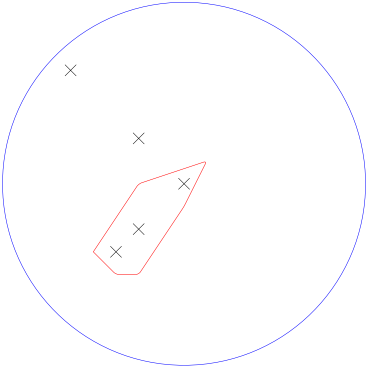
- big enough to contain all the possible solutions
- small enought to avoid unecessary work
  (no redundant solution)
- in a form that is easy to understand
- ...

Goal:
Defining the search space



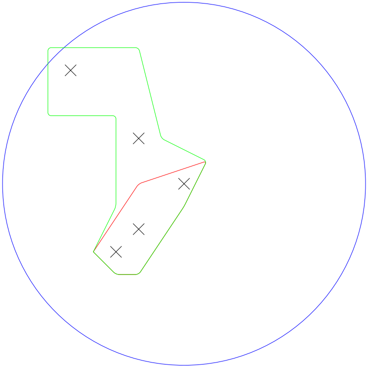
- big enough to contain all the possible solutions
- small enought to avoid unecessary work
  (no redundant solution)
- in a form that is easy to understand
- ...

Goal:
Defining the search space



- big enough to contain all the possible solutions
- small enought to avoid unecessary work
  (no redundant solution)
- in a form that is easy to understand
- . . .

Goal:
Defining the search space



- big enough to contain all the possible solutions
- small enought to avoid unecessary work
  (no redundant solution)
- in a form that is easy to understand
- . . .

1st attempts...

...all the possible logic formulas?
(= all the expressions built connecting the given input nodes with logical connectors)

1st attempts. . .

. . . all the possible logic formulas?
(= all the expressions built connecting the given input nodes with **logical connectors**)

Logical operators and their notation

- $\neg a$: "not a" $\rightarrow$ a has to be False
- $a \vee b$: "a or b"
- $a \wedge b$: "a and b"

- $a \Rightarrow b$: "a implies b"
- $a \Leftrightarrow b$: "equivalent"
- True
- False
- $a \oplus b$: "exclusive or", "xor"
- $a \uparrow b$ "nand"
- $a \downarrow b$: "nor"
- $\ldots$

Logical operators and their notation

- $\neg a$: "not a" $\rightarrow$ a has to be False
- $a \vee b$: "a or b"
- $a \wedge b$: "a and b"

- $a \Rightarrow b \equiv \neg a \vee b$: "$a$ implies b"
- $a \Leftrightarrow b \equiv (\neg a \vee b) \wedge (\neg b \vee a)$: "equivalent"
- True $\equiv a \vee \neg a$
- False $\equiv a \wedge \neg a$
- $a \oplus b \equiv (a \wedge \neg b) \vee (\neg a \wedge b)$: "exclusive or", "xor"
- $a \uparrow b \equiv \neg(a \wedge b)$ "nand"
- $a \downarrow b \equiv \neg(x \vee b)$: "nor"
- $\ldots$

Logical operators and their notation

- $\neg a$: "not a" $\rightarrow$ a has to be False
- $a \vee b$: "a or b"
- $a \wedge b$: "a and b"

abbreviations $\rightarrow$ not absolutely necessary:

- $a \Rightarrow b \equiv \neg a \vee b$: "$a$ implies b"
- $a \Leftrightarrow b \equiv (\neg a \vee b) \wedge (\neg b \vee a)$: "equivalent"
- True $\equiv a \vee \neg a$
- False $\equiv a \wedge \neg a$
- $a \oplus b \equiv (a \wedge \neg b) \vee (\neg a \wedge b)$: "exclusive or", "xor"
- $a \uparrow b \equiv \neg(a \wedge b)$ "nand"
- $a \downarrow b \equiv \neg(x \vee b)$: "nor"
- ...

Logical operators and their notation

functionally complete:

- $\neg a$: "not a" $\rightarrow$ a has to be False
- $a \vee b$: "a or b"
- $a \wedge b$: "a and b"

abbreviations $\rightarrow$ not absolutely necessary:

- $a \Rightarrow b \equiv \neg a \vee b$: "$a$ implies b"
- $a \Leftrightarrow b \equiv (\neg a \vee b) \wedge (\neg b \vee a)$: "equivalent"
- True $\equiv a \vee \neg a$
- False $\equiv a \wedge \neg a$
- $a \oplus b \equiv (a \wedge \neg b) \vee (\neg a \wedge b)$: "exclusive or", "xor"
- $a \uparrow b \equiv \neg(a \wedge b)$ "nand"
- $a \downarrow b \equiv \neg(x \vee b)$: "nor"
- ...

Logical operators and their notation

functionally complete:

- $\neg a$: "not a" $\rightarrow$ a has to be False
- $a \vee b$: "a or b"
- $a \wedge b$: "a and b"

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B}\ \forall i \in V)$$

- "a activates x" $\rightarrow x_{t+1} = f(a_t) = g(a_t, b_t)$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B}\ \forall i \in V)$$

- "a activates x" $\to x_{t+1} = f(a_t) = g(a_t, b_t)$
  $= a_t$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \rightarrow \mathbb{B} \; \forall i \in V)$$

- "a activates x" $\rightarrow$ $x_{t+1} = f(a_t) = g(a_t, b_t)$
  $= a_t \quad = (a_t \wedge b_t) \vee (a_t \wedge \neg b_t)$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B} \ \forall i \in V)$$

- "a activates x" $\to x_{t+1} = f(a_t) = g(a_t, b_t)$
  $= a_t \quad = (a_t \wedge b_t) \vee (a_t \wedge \neg b_t)$
- "both a and b can activate x" $\to x_{t+1} = f(a_t, b_t)$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B}\ \forall i \in V)$$

- "a activates x" $\rightarrow x_{t+1} = f(a_t) = g(a_t, b_t)$
  $= a_t \quad = (a_t \wedge b_t) \vee (a_t \wedge \neg b_t)$
- "both a and b can activate x" $\rightarrow x_{t+1} = f(a_t, b_t) = a_t \vee b_t$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B} \; \forall i \in V)$$

- "a activates x" $\to x_{t+1} = f(a_t) = g(a_t, b_t)$
  $= a_t \quad = (a_t \wedge b_t) \vee (a_t \wedge \neg b_t)$
- "both a and b can activate x" $\to x_{t+1} = f(a_t, b_t) = a_t \vee b_t$

Quick biological examples

in the boolean world:

$$\mathbb{B} = \{0, 1\}$$

a local update function is associated to each agent of the system

$$B = (V, f_i : \mathbb{B}^{k_i \leq n} \to \mathbb{B}\ \forall i \in V)$$

- "a activates x" $\to$ $x \quad = f(a\ ) = g(a\ , b\ )$
  $= a \quad = (a\ \wedge b\ ) \vee (a\ \wedge \neg b\ )$
- "both a and b can activate x" $\to$ $x \quad = f(a\ , b\ ) = a\ \vee b$

Note: from now on, I will omit the time specifications

1st attempt: all the possible logical formulas

Problem:

- For every formula, there is an infinity of other equivalent formula.
  Example:

$$p \quad \equiv \quad p \wedge p \quad \equiv \quad p \wedge p \wedge p \quad \equiv \quad \dots$$

1st attempt: all the possible logical formulas

Problem:

- For every formula, there is an infinity of other equivalent formula.
  Example:

$$p \quad \equiv \quad p \wedge p \quad \equiv \quad p \wedge p \wedge p \quad \equiv \quad \ldots$$

Solution: use a normal form

- modulo some restriction in the synthax, a normal form provide you with a standardized form.
- every formula can be written in an equivalent normal form.
- **two formulas having the same normal form are equivalent**.

  $\rightarrow$ normality is REALLY useful in this context! :D

It exists a lot of normal forms:

- Negation normal form
- Conjunctive normal form
- Disjunctive normal form
- Algebraic normal form
- Prenex normal form
- Skolem normal form
- Blake canonical form
- ...

It exists a lot of normal forms:

- Negation normal form
- Conjunctive normal form
- **Disjunctive normal form**
- Algebraic normal form
- Prenex normal form
- Skolem normal form
- Blake canonical form
- ...

Why using a disjunctive normal form?

Vocabulary + DNF definition

- propositional variable: the basic unit of a formula.
  Examples: a, b.

Vocabulary + DNF definition

- propositional variable: the basic unit of a formula.
  Examples: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Examples: a, $\neg a$

Vocabulary + DNF definition

- propositional variable: the basic unit of a formula.
  Examples: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Examples: a, $\neg a$
- cube (or conjunctive clause): conjunction of litterals.
  Examples: $True$ (conjunction of zero literals), a, $a \wedge a$, $a \wedge b$.

Vocabulary + DNF definition

- propositional variable: the basic unit of a formula.
  Examples: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Examples: a, $\neg a$
- cube (or conjunctive clause): conjunction of litterals.
  Examples: $True$ (conjunction of zero literals), a, $a \wedge a$, $a \wedge b$.
- a formula in disjunctive normal form (DNF) is a disjunction of cubes. Example:

$$(a \wedge b \wedge c) \vee (a \wedge \neg a \wedge b) \vee a$$

Another restriction : satisfiable DNF only

Theorem: formula in DNF form is satifiable if and only if there is at least one cube in which there is not a variable and its negation.

Examples:
a is satisfiable
$a \wedge \neg a$ is not satisfiable
$(a \wedge \neg a) \vee b$ is satisfiable ($\equiv b$).

Why are DNF so useful for us? — Reason n° 1

Because straitforward interpretation for our mind:
a satisfiable DNF is just a list of the combination of inputs where the output happened to be True.

Truth table (from dynamical constraints) :

a	b	$x = f(a, b)$
0	0	0
0	1	0
1	0	0
1	1	1

Why are DNF so useful for us? — Reason n° 1

Because straitforward interpretation for our mind:
a satisfiable DNF is just a list of the combination of inputs where the output happened to be True.

Truth table (from dynamical constraints) :

a	b	$x = f(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

Corresponding DNF :

$$a \wedge b$$

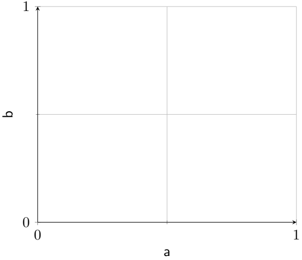
Why are DNF so useful for us? — Reason n° 2

Because easy to represent. $\rightarrow$ hypercubical representation

Truth table of the DNF $a \wedge b$:

a	b	$x = f(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

Its hypercubical representation:



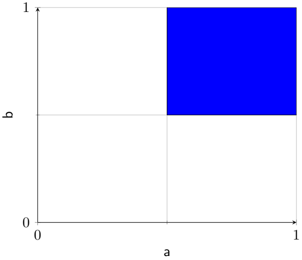
Why are DNF so useful for us? — Reason n° 2

Because easy to represent. $\rightarrow$ hypercubical representation

Truth table of the DNF $a \wedge b$:

a	b	$x = f(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

Its hypercubical representation:

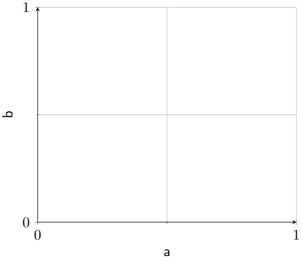


Hypercubical representation: another example

Truth table of the DNF $(a \wedge \neg b) \vee (\neg a \wedge b)$ (xor):

a	b	$x = f(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

Its hypercubical representation:

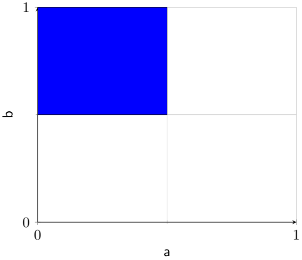


Hypercubical representation: another example

Truth table of the DNF $(a \wedge \neg b) \vee (\neg a \wedge b)$ (xor):

a	b	$x = f(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

Its hypercubical representation:

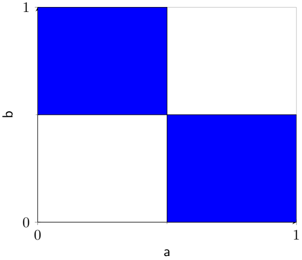


Hypercubical representation: another example

Truth table of the DNF $(a \wedge \neg b) \vee (\neg a \wedge b)$ (xor):

a	b	$x = f(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

Its hypercubical representation:



2nd attempt: use DNF

Can 2 different DNF formula be equivalent?

Yes! Their difference can be in the order of litterals or the order of cubes:

$$
\begin{aligned}
&(a \land b) \lor (\neg c \land \neg b) \\
\equiv &(b \land a) \lor (\neg c \land \neg b) \\
\equiv &(\neg c \land \neg b) \lor (b \land a)
\end{aligned}
$$

3rd attempt:

Can 2 different DNF formula with more difference that simply the order of cubes of the order of litterals be equivalent?

Yes! Because of the redundancy of subsumed cubes:

$$(a \wedge b) \quad \equiv \quad (a \wedge b) \vee (a \wedge b \wedge c)$$

Subsomption, kézako ?

A cube c_1 subsumes a cube c_2 if all the literals in c_1 are in c_2.

A cube c_1 subsumes a cube c_2 if c_1 is "included" in c_2, putting aside the order of litteral and the eventual repetition.

- $a \wedge \neg b$ subsumes $a \wedge \neg b \wedge c$
- $a \wedge \neg b$ subsubes $a \wedge \neg b \wedge c \wedge c$.

Redundancy of subsumed clause

Subsumed cubes are redundant.
Let $c_1, \ldots c_n$. If c_j subsumes c_i, for $i \neq j$, then:
$c_1 \vee \ldots \vee c_{i-1} \vee c_i \vee c_{i+1} \vee \ldots \vee c_n \equiv c_1 \vee \ldots \vee c_{i-1} \vee c_{i+1} \vee \ldots \vee c_n$

Example:

$$(a \wedge \neg b) \quad \vee(\neg a \wedge b) \quad \vee(\underbrace{a \wedge \neg b} \wedge c \wedge \neg d)$$
$$\equiv(a \wedge \neg b) \quad \vee(\neg a \wedge b)$$

4th attempt:

Can 2 different DNF formula with more difference than simply the order of cubes or the order of literals, and where none of the cube subsume another cube in the same formula be different?

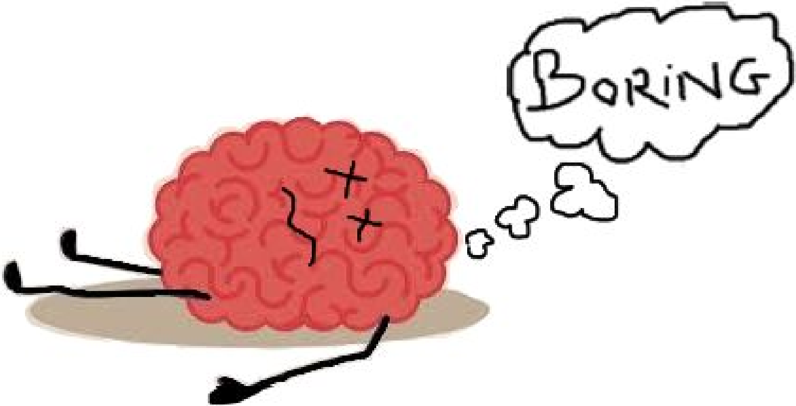
Yes! For example, both:

$$\neg a \vee (a \wedge \neg b) \text{ and } \neg b \vee (b \wedge \neg a)$$

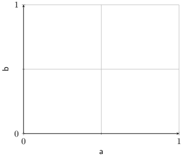
are equivalent to

$$(\neg a \wedge b) \;\vee (\neg a \wedge \neg b) \vee (a \wedge \neg b)$$

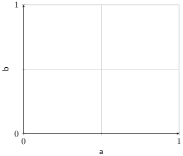
Formal proof? No, thanks...



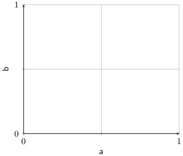
Visual intuition of the proof ? Yes, please: :D



$$\neg x \vee (x \wedge \neg y)$$

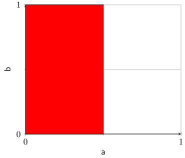


$$\neg y \vee (\neg x \wedge y)$$

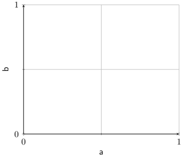


$$\neg x \wedge \neg y \vee (x \wedge \neg y) \vee (\neg x \wedge y)$$

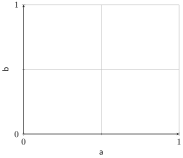
Visual intuition of the proof ? Yes, please: :D



$$\neg x \vee (x \wedge \neg y)$$

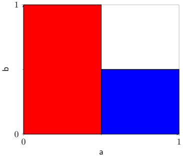


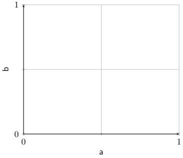
$$\neg y \vee (\neg x \wedge y)$$

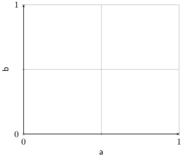


$$\neg x \wedge \neg y \vee (x \wedge \neg y) \vee (\neg x \wedge y)$$

Visual intuition of the proof ? Yes, please: :D





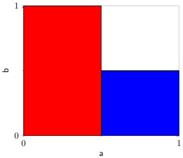


$$\neg x \lor (x \land \neg y)$$

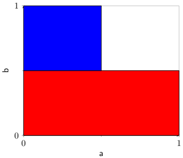
$$\neg y \lor (\neg x \land y)$$

$$\neg x \land \neg y \lor (x \land \neg y) \lor (\neg x \land y)$$

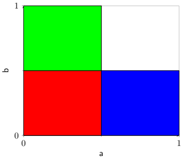
Visual intuition of the proof ? Yes, please: :D



$$\neg x$$
$$\vee(x \wedge \neg y)$$



$$\neg y$$
$$\vee(\neg x \wedge y)$$



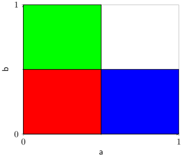
$$\neg x \wedge \neg y$$
$$\vee(x \wedge \neg y)$$
$$\vee(\neg x \wedge y)$$

Their is another...

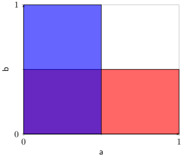
... DNF formula that is equivalent. And even shorter!

$$\neg x \wedge \neg y$$

Illustration



$$\neg a \wedge \neg b$$
$$\vee(a \wedge \neg b)$$
$$\vee(\neg a \wedge b)$$



$$\neg b$$
$$\vee(\neg a)$$

About minimal DNF

Minimal DNF: smallest DNF among all the equivalent DNFs.

About minimal DNF

Minimal DNF: **smallest** DNF among all the equivalent DNFs.

Length of a formula: the number of litteral
Example:
$(a \wedge \neg b) \vee (a \wedge c)$: length $= 4$
$a \wedge a \wedge a \wedge a \wedge a$: length $= 5$

About minimal DNF

Minimal DNF: **smallest** DNF among all the equivalent DNFs.

Length of a formula: the number of litteral
Example:
$(a \wedge \neg b) \vee (a \wedge c)$: length $= 4$
$a \wedge a \wedge a \wedge a \wedge a$: length $= 5$

Example of min DNF:
a is a minimal DNF, but $a \wedge a$ is not.

5th attempt: min DNF

5th attempt: min DNF

Are min DNF unique, modulo order of literals and order of cubes?

5th attempt: min DNF

Are min DNF unique, modulo order of literals and order of cubes?

mmh... yes and no

5th attempt: min DNF

Are min DNF unique, modulo order of literals and order of cubes?

mmh... yes and **no**
Example: There is several way of minimizing this DNF:

$$(a \wedge \neg b) \vee (\neg a \wedge b) \vee (b \wedge \neg c) \vee (\neg b \wedge c)$$

- $(a \wedge \neg b) \vee (b \wedge \neg c) \vee (\neg a \wedge c)$
- $(\neg a \wedge b) \vee (\neg b \wedge c) \vee (a \wedge \neg c)$

5th attempt: min DNF

Are min DNF unique, modulo order of literals and order of cubes?

mmh... **yes** and no
Example: There is several way of minimizing this DNF:

$$(a \wedge \neg b) \vee (\neg a \wedge b) \vee (b \wedge \neg c) \vee (\neg b \wedge c)$$

- $(a \wedge \neg b) \vee (b \wedge \neg c) \vee (\neg a \wedge c)$
- $(\neg a \wedge b) \vee (\neg b \wedge c) \vee (a \wedge \neg c)$

But not the same biological interpretation

Recipie to generate all the possible min DNF of n boolean variables

Recipie to generate all the possible min DNF of n boolean variables

1. Use the topological constraint to have the set of possible inputs

$$\{a, \neg a, b, \neg b, c, \neg c\} \to \{a, b, c\} = E$$

Recipie to generate all the possible min DNF of n boolean variables

1. Use the topological constraint to have the set of possible inputs
$$\{a, \neg a, b, \neg b, c, \neg c\} \to \{a, b, c\} = E$$
2. Get all the possible cubes $\to$ compute the powerset of E:

$$\mathscr{P}(E) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, E\}$$

$$|\mathscr{P}(E)| = 2^{|E|} = 2^3 = 8$$

(powerset = the set of all possible subsets)

Recipie to generate all the possible min DNF of n boolean variables

1. Use the topological constraint to have the set of possible inputs
$$\{a, \neg a, b, \neg b, c, \neg c\} \rightarrow \{a, b, c\} = E$$
2. Get all the possible cubes $\rightarrow$ compute the powerset of E:
$$\mathscr{P}(E) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, E\}$$
$$|\mathscr{P}(E)| = 2^{|E|} = 2^3 = 8$$
3. Remove the impossible cubes : $l \wedge \neg l$

Recipie to generate all the possible min DNF of n boolean variables

1. Use the topological constraint to have the set of possible inputs
$$\{a, \neg a, b, \neg b, c, \neg c\} \rightarrow \{a, b, c\} = E$$
2. Get all the possible cubes $\rightarrow$ compute the powerset of E:
$$\mathscr{P}(E) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, E\}$$
$$|\mathscr{P}(E)| = 2^{|E|} = 2^3 = 8$$
3. Remove the impossible cubes : $l \wedge \neg l$
4. Order (partially) the cubes by their inclusion : $a \subset ab \subset abc$

Recipie to generate all the possible min DNF of n boolean variables

1. Use the topological constraint to have the set of possible inputs
$$\{a, \neg a, b, \neg b, c, \neg c\} \to \{a, b, c\} = E$$
2. Get all the possible cubes $\to$ compute the powerset of E:
$$\mathscr{P}(E) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, E\}$$
$$|\mathscr{P}(E)| = 2^{|E|} = 2^3 = 8$$
3. Remove the impossible cubes : $l \wedge \neg l$
4. Order (partially) the cubes by their inclusion : $a \subset ab \subset abc$
5. Enumerate all the antichains

(antichain = subset such that any two distinct element are incomparable)

$$
\begin{tikzcd}
 & empty \arrow[dl] \arrow[d] \arrow[dr] & \\
a \arrow[d] \arrow[dr] & b \arrow[dl] \arrow[dr] & c \arrow[dl] \arrow[d] \\
a, b \arrow[dr] & a, c \arrow[d] & b, c \arrow[dl] \\
 & a, b, c &
\end{tikzcd}
$$

$$\begin{tikzcd}
 & empty \arrow[dl] \arrow[d] \arrow[dr] & \\
a \arrow[d] \arrow[dr] & -a \arrow[dl] \arrow[dr] & b \arrow[dl] \arrow[d] \\
a, -a \arrow[dr] & a, b \arrow[d] & -a, b \arrow[dl] \\
 & a, -a, b &
\end{tikzcd}$$

State of our current ASP implementation

State of our current ASP implementation

- Difficulties to encode the antichain relation

State of our current ASP implementation

- Difficulties to encode the antichain relation

Workarround:

- some minimisation rules encoded in ASP.
- the search space is all the DNF that pass these filters.
- at the end, we minimise the solutions with an external program, and remove redundant solutions.
  - Famous exact algorithm from Quine and McCluskey
  - Famous heuristic from Berckley University : Expresso

State of our current ASP implementation

- Difficulties to encode the antichain relation

Workarround:

- some minimisation rules encoded in ASP.
- the search space is all the DNF that pass these filters.
- at the end, we minimise the solutions with an external program, and remove redundant solutions.
  - Famous exact algorithm from Quine and McCluskey (broken in Python)
  - Famous heuristic from Berckley University : Expresso

Thanks for your attention. . .
Any questions? :)

+ thanks to Justine and Alexandre for having discuss the link between DNF and partially ordered sets

Annexes

References I

Poly de cours "outils logiques" par Ralf Treinen
https://www.irif.fr/~kesner/enseignement/ol3/poly.pdf

On non unicity of min DNF : https://math.stackexchange.com/questions/321285/is-there-a-unique-minimal-expression-for-every-boolean-function/321326

normal form

disjunctive normal form

minimal disjunctive normal form

min DNF not unique

There is several way of minimizing this DNF:

$$(\neg a \land b) \lor (a \land \neg b) \lor (a \land b \land c)$$

- $(\neg a \land b) \lor (a \land \neg b) \lor (a \land c)$
- $(\neg a \land b) \lor (a \land \neg b) \lor (b \land c)$
- $(\neg a \land b) \lor (a \land \neg b) \lor (a \land b)$

Functional completness of logical operators

A set S of logical connector is functionaly complete if for all natural number n, and all function f:

$$f : \underbrace{\{0,1\} \times ... \times \{0,1\}}_{n \text{ times}} \rightarrow \{0,1\}$$

one can find a propositional formula p containing only the logical connectors from S, such that p "rzalize" f and $V(p) \subseteq \{x_1, \ldots, x_n\}$, i.e. such that:

???

for all boolean values $b_1, \ldots, b_n \in \{0,1\}$

Functional completness of logical operators

A set of operator is functionaly complete if it allows the expression of all the possible boolean formulas.

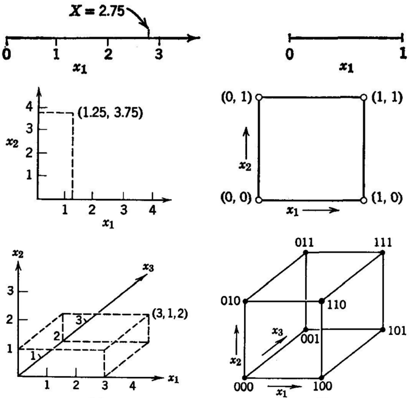
Examples:

- $\{\neg, \wedge, \vee\}$
- $\{\neg, \wedge\}$
- $\{\neg, \vee\}$
- $\{\neg, \Leftrightarrow\}$
- $\{\uparrow\}$

But not:

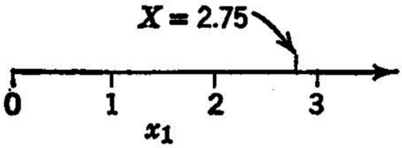
- $\{\wedge, \vee\}$
- $\{\Leftrightarrow\}$

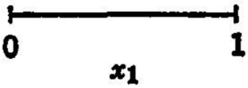
Hypercubical representation



Figures stolen from chapter 7 (Tabular minimization and multiple output circuits) of Introduction to switching theory and logical design, by Hill and Peterson.
`https://archive.org/details/IntroductionToSwitchingTheoryLogicpage/n149`

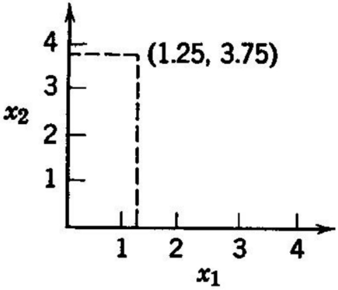
Hypercubical representation

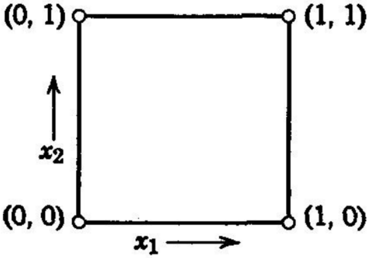




Figures stolen from chapter 7 (Tabular minimization and multiple output circuits) of Introduction to switching theory and logical design, by Hill and Peterson.
`https://archive.org/details/IntroductionToSwitchingTheoryLogicpage/n149`

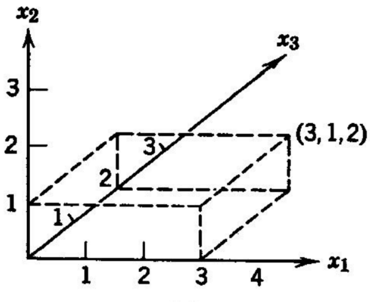
Hypercubical representation

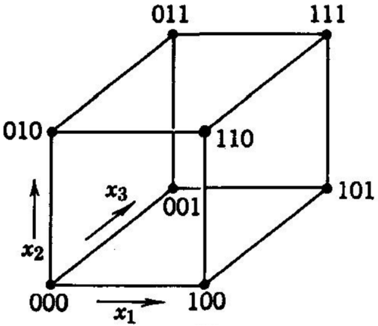




Figures stolen from chapter 7 (Tabular minimization and multiple output circuits) of Introduction to switching theory and logical design, by Hill and Peterson.
`https://archive.org/details/IntroductionToSwitchingTheoryLogicpage/n149`

Hypercubical representation





Figures stolen from chapter 7 (Tabular minimization and multiple output circuits) of Introduction to switching theory and logical design, by Hill and Peterson.
`https://archive.org/details/IntroductionToSwitchingTheoryLogicpage/n149`

Length explosion

The transformation of a formula in its DNF form can make the length grow exponentially.

For example, the DNF of the formula:

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2)$$

is:

$$\begin{aligned} &(x_1 \wedge x_2) \\ \vee&(x_1 \wedge y_2) \\ \vee&(y_1 \wedge x_2) \\ \vee&(y_1 \wedge y_2) \end{aligned}$$

$$(x1 \vee y1) \wedge (x2 \vee y2) \wedge (x3 \vee y3)$$

is:

$$\begin{aligned} &(x1 \wedge x2 \wedge x3) \\ \vee&(x1 \wedge x2 \wedge y3) \\ \vee&(x1 \wedge y2 \wedge x3) \\ \vee&(x1 \wedge y2 \wedge y3) \\ \vee&(y1 \wedge x2 \wedge x3) \\ \vee&(y1 \wedge x2 \wedge y3) \\ \vee&(y1 \wedge y2 \wedge x3) \\ \vee&(y1 \wedge y2 \wedge y3) \end{aligned}$$

Length explosion — generality

Generally, puting a formula of the form:

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \ldots \wedge (x_n \vee y_n)$$

in DNF gives a formula with 2^n cubes, each cubes having n litterals.

Neutral elements

The conjunction of zero formula is a neutral element and gives True.

- Adding nothing to a conjunction should give us an equivalent formula
- Analogy with the sum of no numbers is 0: $\sum_{i=1}^{i=0} i = 0$

The disjunction of zero formula is a neutral element and gives False.

Vocabulary + DNF definition — version 1

- propositional variable: the basic unit of a formula.
  Example: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Example: a, not a, b, not b
- cube (or conjunctive clause): either the True constant,
  or the conjunction between at leat two litterals. Example:
  True, a, a and a, a and b.
- a formula in disjunctive normal form (DNF) is either:
  the constant False, or a cube, or a disjunction of at least two
  cubes. Example:

$$(a \wedge b \wedge c) \vee (a \wedge \neg a \wedge b) \vee a$$

Vocabulary + DNF definition — version 2

- propositional variable: the basic unit of a formula.
  Example: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Example: a, not a, b, not b
- cube (or conjunctive clause): conjunction of zero of more litterals. Example: True, a, a and a, a and b.
- a formula in disjunctive normal form (DNF) is either:
  a disjunction of zero or more cubes. Example:

$$(a \wedge b \wedge c) \vee (a \wedge \neg a \wedge b) \vee a$$

Vocabulary + DNF definition — version 3

- propositional variable: the basic unit of a formula.
  Example: a, b.
- litteral: either a propositional variable,
  or the negation of a propositional variable.
  Example: a, not a, b, not b
- cube (or conjunctive clause): conjunction of litterals.
  Example: True, a, a and a, a and b.
- a formula in disjunctive normal form (DNF) is either:
  a disjunction of cubes. Example:

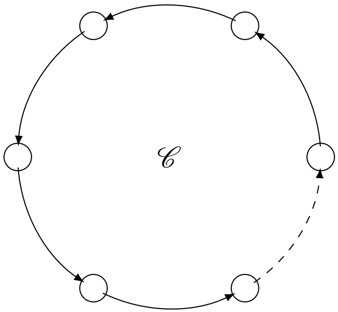
$$(a \wedge b \wedge c) \vee (a \wedge \neg a \wedge b) \vee a$$

About constraints

logical function synthesis, using constraints about

- the definition domain of the function (hard constraint)
  "perhaps x activated by a, or perhaps by b, or perhaps both"
  $\rightarrow \quad x_{t+1} = f(a_t); \quad x_{t+1} = g(b_t); \quad x_{t+1} = h(a_t, b_t)$
- $+$ specification abouts their behavior (soft constraint to be optimized).
  TODO : explain why soft (orally ?)

Trajectory and attractors (cycles and fixed points)



Methods from which I can get inspired

boolean function synthesis from biological timecourse data (and a prior knowledge network)

- T. Akutsu, S. Miyano, S. Kuhara Identification of genetic networks from a small number of gene expression patterns under the Boolean network model Pacific Symposium on Biocomputing (1999)
- REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. Liang et al. 1998
- Best-fit extension Lähdesmäki et al 2003

Quick biological examples

- $x_{t+1} = f(a_t) = a_t$
  a is an input that determine the status of x. The verity value of the formula is the status of x at the next time step.
- $x_{t+1} = f(a_t, b_t) = a_t \vee b_t$
  a and b are the inputs that determine the status of x. The verty value of the formula is the status of x at the next time step.

Quick biological examples

- $x \quad = f(a\ ) = a$
  a is an input that determine the status of x. The verity value of the formula is the status of x at the next time step.
- $x \quad = f(a\ , b\ ) = a\ \vee b$
  a and b are the inputs that determine the status of x. The verty value of the formula is the status of x at the next time step.

Note: from now on, I will omit the time specifications

About search spaces

10^{80} : number of atom in the universe
10^{20} : number of combination it is commonly assumed to be able to enumerate per year on modern hardware

About search spaces

10^{80} : number of atom in the universe
10^{20} : number of combination it is commonly assumed to be able to enumerate per year on modern hardware

- Base search space : the number of combinations that each Solution model is able to represent, regardless if those Solutions are feasible or infeasible ($=$ have broken hard constraints)
- Feasible search space : all infeasible solutions (the ones that breaks at least one hard constraints) are discarded.

You already know: the given constraints concern definition domain and state sequence.
Is it enought ?

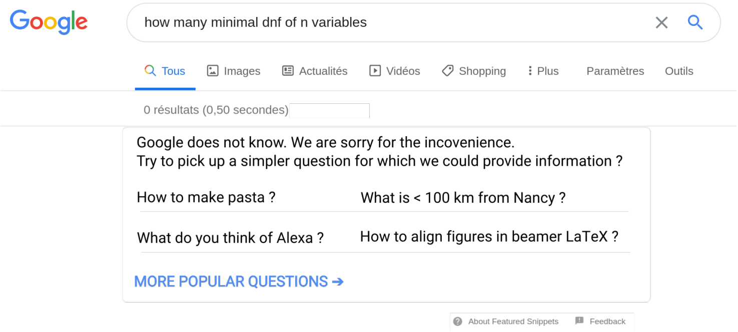
How many min DNF?

muddle through enumeration by hand:

- min DNF of length 0: 2 (True, False)
- min DNF of length 1: 2 (a, $\neg a$)
- min DNF of length 2:

Why do I want to talk about this?

1. because google sadly does not have an obvious answer



Why do I want to talk about this?

1. because google sadly does not have an obvious answer
2. because I would like to ask the question to people who might know, but I first need to order my thought about it

Why do I want to talk about this?

1. because google sadly does not have an obvious answer
2. because I would like to ask the question to people who might know, but I first need to order my thought about it
3. because some of you like hard problems, and could come up with some useful insights

Why do I want to talk about this?

1. because google sadly does not have an obvious answer
2. because I would like to ask the question to people who might know, but I first need to order my thought about it
3. because some of you like hard problems, and could come up with some useful insights
4. because if would know it, I would have a more precise idea about the complexity of the approach I would like to use for my main thesis project