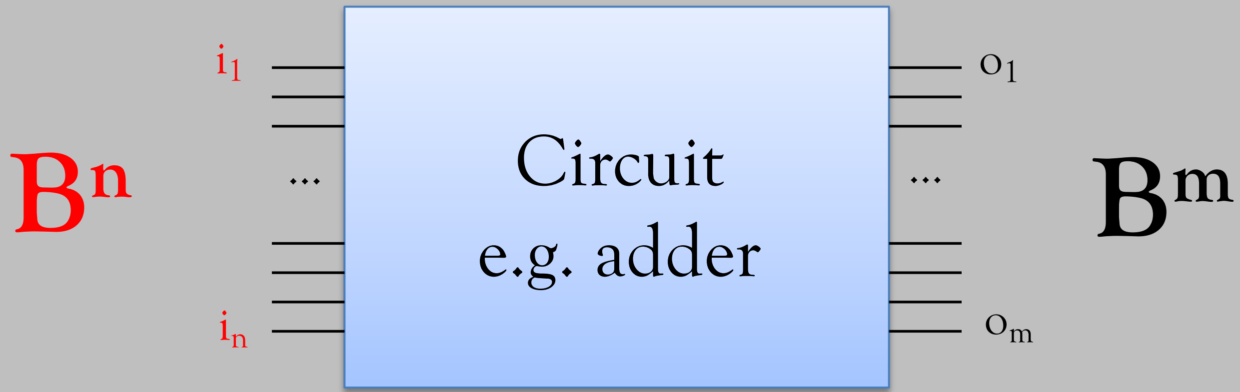
The Boolean Calculus:
Boolean Functions,
Boolean Algebras,
Boolean Expressions

Becker/Molitor, Chapter 2
Harris/Harris, Chapters 2.2 and 2.3

Jan Reineke
Universität des Saarlandes

Boolean functions as a mathematical model for circuits

Due to binary representation of numbers and characters, assume $B=\{0,1\}$.



Circuit implements/computes a function $f : B^n \rightarrow B^m$

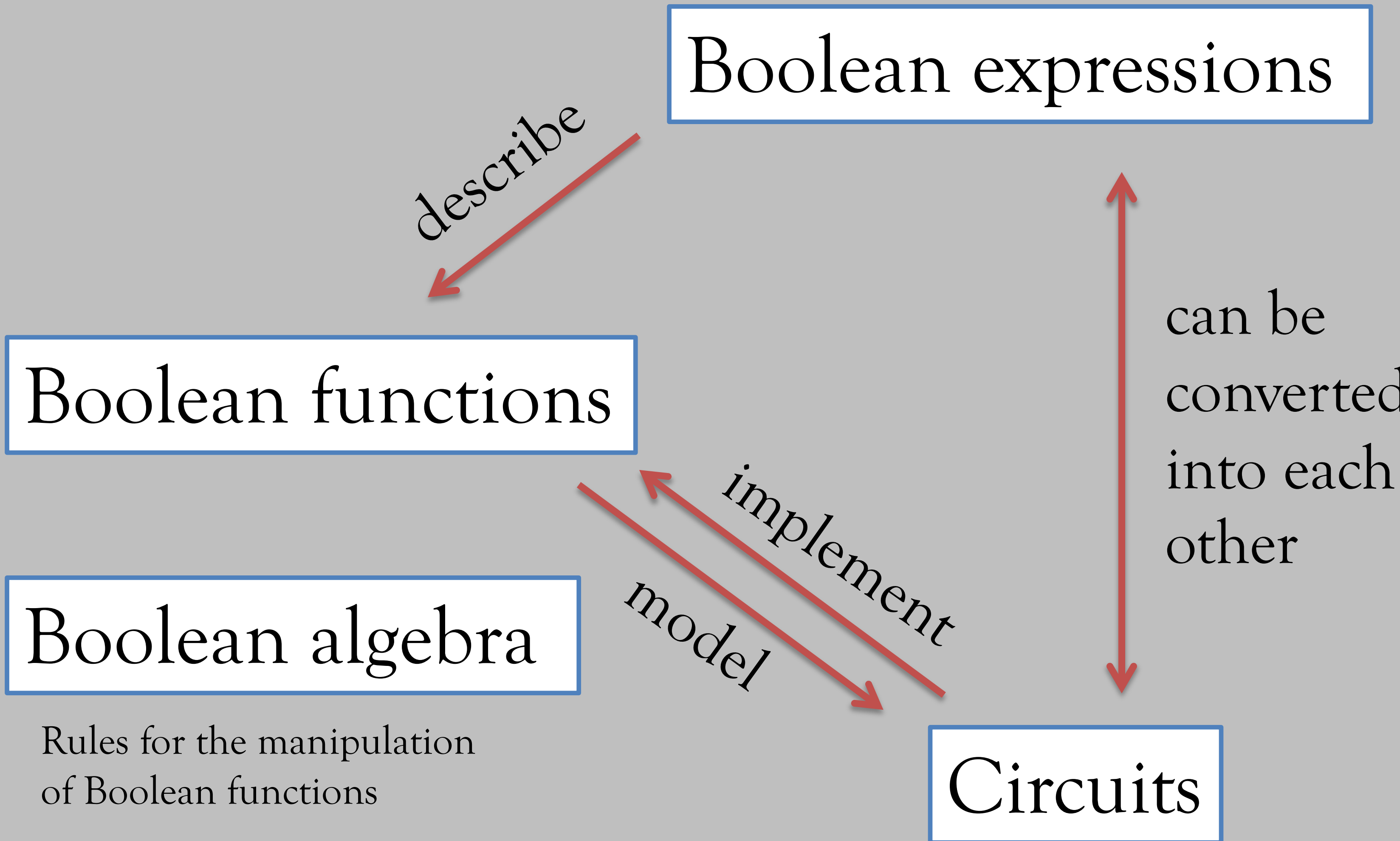
Central questions

1. Can every Boolean function be implemented by some circuit?

2. Given a Boolean function, can we systematically construct a circuit that implements this function?

3. Given a Boolean function, can we systematically construct an **efficient** circuit that implements this function?

Overview


Boolean expressions
describe
Boolean functions
implement
model
can be converted into each other
Boolean algebra
Rules for the manipulation of Boolean functions
Circuits

Boolean functions

- A mapping $f : \mathbf{B}^n \rightarrow \mathbf{B}^m$ is called **(total) Boolean function** in n variables.

- $\mathbf{B}_{n,m} := \mathbf{B}^n \rightarrow \mathbf{B}^m$

- A mapping $f : D \rightarrow \mathbf{B}^m$ with $D \subseteq \mathbf{B}^n$ is called **(partial) Boolean function** in n variables.
  $\mathbf{B}_{n,m}(D) := D \rightarrow \mathbf{B}^m$ for $D \subseteq \mathbf{B}^n$

Truth tables

Boolean functions can be represented via **truth tables**:

x_1	x_2	x_3	s_1	s_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2^n input combinations (columns x_1, x_2, x_3)

result vector (columns s_1, s_0)

Question: How many Boolean functions $\mathbf{B}_{n,m}$ are there?

$|\mathbf{B}_{n,m}| = 2^{m*2^n}$

On-Set and Off-Set

Let $m = 1$, then

- **ON**(f) $:= \{\alpha \in B^n \mid f(\alpha) = 1\}$ is the **On-Set** of f,
- **OFF**(f)$:=\{\alpha \in B^n \mid f(\alpha) = 0\}$ is the **Off-Set** of f.

For $f : D \to B^m$ with $D \subseteq B^n$ we call

- the set $def(f) := D$ **domain (of definition)** of f,
- the set $DC(f) := B^n \setminus D$ **don't care set** of f.

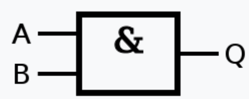
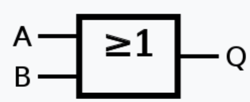
Logic gates implement simple Boolean functions

Electronic switches that implement Boolean functions

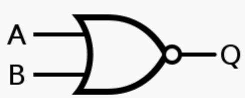
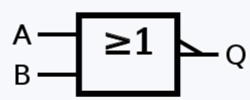
are constructed from simple electronic components (transistors)

(*later*: more about their construction)

[Source: https://en.wikipedia.org/wiki/Logic_gate]

<table>
<tr><th colspan="5">Conjunction and Disjunction</th></tr>
<tr>
<td>AND</td>
<td></td>
<td></td>
<td>$A \cdot B$ or $A \wedge B$</td>
<td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>Q</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td>$A + B$ or $A \vee B$</td>
<td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><td>A</td><td>B</td><td>Q</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
</table>

◆ ◆ ◆

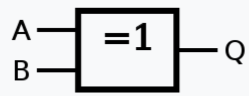
<table>
<tr><th colspan="5">Alternative denial and Joint denial</th></tr>
<tr>
<td>NAND</td>
<td>
</td>
<td>
</td>
<td>$\overline{A \cdot B}$ or $A \uparrow B$</td>
<td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>Q</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
<tr>
<td>NOR</td>
<td>
</td>
<td>
</td>
<td>$\overline{A + B}$ or $A \downarrow B$</td>
<td>
<table>
<tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr>
<tr><th>A</th><th>B</th><th>Q</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
</table>

[Source: https://en.wikipedia.org/wiki/Logic_gate]

• • •

Exclusive or and Biconditional

XOR

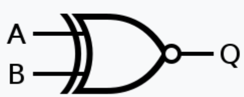


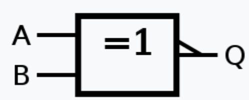
$A \oplus B$ or $A \veebar B$

INPUT		OUTPUT
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

The output of a two input exclusive-OR is true only when the two input values are *different*, and false if they are equal, regardless of the value. If there are more than two inputs, the output of the distinctive-shape symbol is undefined. The output of the rectangular-shaped symbol is true if the number of true inputs is exactly one or exactly the number following the "=" in the qualifying symbol.

XNOR





$\overline{A \oplus B}$ or $A \odot B$

INPUT		OUTPUT
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

[Source: https://en.wikipedia.org/wiki/Logic_gate]

Two-element Boolean algebra

- Dates back to Boole's logical calculus (1847)

- Two elements: **B** = {0, 1}
- Two binary operators:
  - Conjunction, logical and:
    ∧ (also ·, AND)
  - Disjunction, logical or
    ∨ (also +, OR)
- One unary operator:
  - Negation, **¬ (also ~, NOT, `)**

George Boole (1815-1864)

Two-element Boolean algebra

We consider $\mathbf{B} = \{0,1\}$ with the two binary operators

- $\wedge$ (Conjunction),
- $\vee$ (Disjunction), and
- the unary operator $\neg$ (Negation).

Which laws hold for $\mathbf{B}$ under these operators?

Boolean algebras

- **Boolean algebra** =
  Algebraic structure with particular properties
- Let M be a set equipped with binary operators $\cdot$ and $+$ and a unary operator $\sim$ are defined.
- The tuple $(M, \cdot, +, \sim)$ is called **Boolean algebra**,
  if M is a non-empty set and for all $x, y, z \in M$
  the following axioms hold:

Commutativity	$x+y=y+x$	$x\cdot y=y\cdot x$
Associativity	$x+(y+z)=(x+y)+z$	$x\cdot(y\cdot z)=(x\cdot y)\cdot z$
Absorption	$x+(x\cdot y)=x$	$x\cdot(x+y)=x$
Distributivity	$x+(y\cdot z)=(x+y)\cdot(x+z)$	$x\cdot(y+z) =(x\cdot y)+(x\cdot z)$
Complements	$x+(y\cdot(\sim y))=x$	$x\cdot(y+ (\sim y))= x$

Theorem: $(\mathbb{B}, \wedge, \vee, \neg)$ is a Boolean algebra.

Further laws in Boolean algebras

- There are further laws
  that *follow* from these axioms.

- Before considering such laws and their proofs:
  Examples of other Boolean Algebras

Boolean algebra of Boolean functions in *n* variables

$\mathbf{B}_n := \mathbf{B}_{n,1}$	Set of **Boolean functions** in *n* variables, m=1		
$f \cdot g \in \mathbf{B}_n$	defined as	$(f \cdot g)(\alpha) = f(\alpha) \cdot g(\alpha)$	$\forall \alpha \in \mathbf{B}^n$
$f + g \in \mathbf{B}_n$	defined as	$(f + g)(\alpha) = f(\alpha) + g(\alpha)$	$\forall \alpha \in \mathbf{B}^n$
$\sim f \in \mathbf{B}_n$	defined as	$(\sim f)(\alpha) = \sim(f(\alpha))$	$\forall \alpha \in \mathbf{B}^n$

Operators in the Boolean algebra of Boolean functions

Operators in the Two-element Boolean algebra

Theorem: $(B_n, \cdot , + , \sim)$ is a Boolean Algebra.

Proof: Showing that all axioms hold.

Boolean algebra of subsets of S

- S : arbitrary non-empty set
- 2^S : the power set of S
- $M_1 \cup M_2$: the union of the sets M_1 and M_2 from 2^S
- $M_1 \cap M_2$: the intersection of the sets M_1 and M_2 from 2^S
- $\sim M$: the complement $S \setminus M$ of M relative to S

Theorem: $(2^S, \cap, \cup, \sim)$ is a Boolean algebra.

Proof: Showing that all axioms hold.

Further laws in Boolean algebras, derivable from the axioms

- **Existence of neutral (identity) elements:**

  $\exists 0 : \forall x : x + 0 = x, x \cdot 0 = 0$

  $\exists 1 : \forall x : x \cdot 1 = x, x + 1 = 1$

- **Double negation:**

  $\forall x : \sim(\sim x) = x$

- **Uniqueness of complements:**

  $\forall x,y : (x \cdot y = 0 \text{ and } x + y = 1) \Rightarrow \quad y = \sim x$

- **Idempotence:**

  $\forall x : x + x = x \qquad \forall x : x \cdot x = x$

- **de Morgan's laws:**

  $\forall x,y : \sim (x + y) = (\sim x) \cdot (\sim y) \qquad \forall x,y : \sim (x \cdot y) = (\sim x) + (\sim y)$

- **Consensus law:**

  $\forall x,y,z : (x \cdot y) + ((\sim x) \cdot z) = (x \cdot y) + ((\sim x) \cdot z) + (y \cdot z)$

  $\forall x,y,z : (x + y) \cdot ((\sim x) + z) = (x + y) \cdot ((\sim x) + z) \cdot (y + z)$

Proof (Idempotence):

$x \overset{\text{Absorption}}{=} x + (x \cdot (y + \sim y)) \overset{\text{Complements}}{=} x + x$

Proof (Neutr. elements):

Let $0 = x \cdot \sim x$

Then we have:

$x + 0 = x + (x \cdot \sim x) \overset{\text{Complements}}{=} x$

Duality principle of Boolean algebra

Duality principle

Let p be an arbitrary law of Boolean algebra, then the dual of p is also a law of Boolean algebra.

The dual of p, is obtained from p by exchanging $+$ and $\cdot$, as well as 0 and 1.

Example

$$(x \cdot y) + ((\sim x) \cdot z) + (y \cdot z) = (x \cdot y) + ((\sim x) \cdot z)$$
$$(x + y) \cdot ((\sim x) + z) \cdot (y + z) = (x + y) \cdot ((\sim x) + z)$$

Boolean expressions: Goals

- *Wanted*: A way to describe Boolean functions
- *So far*: Truth tables. However: for n variables 2^n entries!

- *Goals*:
  - Enable compact representation
  - Synthesis of circuits

Boolean expressions

- Let $X_n = \{x_1, x_2, \ldots, x_n\}$ be the set of variables.
- **Boolean expressions** are defined on the alphabet
  $A = X_n \cup \{0, 1 , + , \cdot , \sim , (,)\}$,
  i.e. Boolean expressions are a subset of A^*.

Boolean expressions

Definition:

The set $BE(X_n)$ of fully parenthesized Boolean expressions over X_n is the smallest subset of A^*, inductively defined as follows:

- The elements 0 and 1 are Boolean expressions
- The variables $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are Boolean expressions
- Let **g** and **h** be Boolean expressions. Then so is their Disjunction $(\mathbf{g} + \mathbf{h})$, their Conjunction $(\mathbf{g} \cdot \mathbf{h})$, and their Negation $(\sim\mathbf{g})$.

BE(X_n): Operator precedence

- Negation $\sim$ precedes conjunction $\cdot$
- Conjunction $\cdot$ precedes disjunction $+$

→ Parentheses can be omitted without introducing ambiguities

Instead of $\cdot$ we often write $\wedge$,

instead of $+$ also $\vee$,

instead of $\sim x_i$ also x_i' or

Example:

$$\sim x_1 \cdot x_2 + x_3 \equiv ((\sim x_1) \cdot x_2) + x_3$$

Interpretation of Boolean expressions

- Every Boolean expression can be associated with a Boolean function via an interpretation function $\psi : BE(X_n) \rightarrow \mathbf{B}_n$.
- ψ is defined inductively as follows:
  - $\psi(0) = \mathbf{0} = \lambda x_1, \ldots, x_n.\ 0$
  - $\psi(1) = \mathbf{1} = \lambda x_1, \ldots, x_n.\ 1$
  - $\psi(x_i)(\alpha_1,\ldots,\alpha_n) = \alpha_i \quad \forall \alpha \in \mathbf{B}^n$ ("projection")
  - $\psi((g+h)) = \psi(g) + \psi(h)$ ("disjunction")
  - $\psi((g \cdot h)) = \psi(g) \cdot \psi(h)$ ("conjunction")
  - $\psi((\sim g)) = \sim(\psi(g))$ ("negation")

Elements of the alphabet

Operators of the Boolean alg. of Boolean functions

Interpretation of Boolean expressions

- For a valuation $\alpha \in \mathbf{B}^n$, $\psi(e)(\alpha)$ is obtained by replacing $\mathbf{x_i}$ by α_i for all i in $\mathbf{e}$ and evaluation in the Boolean algebra $\mathbf{B}$.

- Two BEs $\mathbf{e_1}$ and $\mathbf{e_2}$ are called **equivalent** ($\mathbf{e_1} \equiv \mathbf{e_2}$) *if and only if* $\psi(e_1) = \psi(e_2)$.

For instance, we have $x_1 \equiv x_1 + x_1$

Proof: $\psi(x_1) = \psi(x_1) + \psi(x_1) = \psi(x_1+x_1)$

Idempotence

Definition ψ

Boolean functions versus Boolean expressions

- Let $\psi(e)=f$ for a Boolean expression e and a Boolean function f. Then we say
  - that e is a **Boolean expression** for f, and
  - that e **describes** the Boolean function f.

Every Boolean expression describes some Boolean function.

But can every Boolean function be described by some Boolean expression?

Systematic construction of Boolean expressions

Brainstorming:

How to "build" a Boolean expression for an arbitrary Boolean function defined by a truth table?

x_1	x_2	x_3	s
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Special Boolean expressions: Literals and monomials

- The Boolean expressions x_i and x_i' are called literals, where
  - x_i is a positive literal and
  - x_i' is a negative literal .
- A monomial (also product) is
  - a conjunction of literals with additional properties:
    - every literal appears at most once,
    - it does not contain both the positive and the negative literal of any variable.
  - or it is the Boolean expression **1**.
- A monomial is called minterm, if each variable occurs either as positive or as negative literal.

Question: What kind of functions are described by minterms (and more generally monomials)?

Contruction of Boolean expressions from truth tables

1. Consider all rows for which the function is 1.

1. Construct the minterm for the valuation of x_1, x_2 und x_3 in the row as follows:
   - if x_i is 1 $\Rightarrow x_i$
   - if x_i is 0 $\Rightarrow x_i'$

2. Combine all minterms by a disjunction

x_1	x_2	x_3	s
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Special Boolean expressions: Polynomials

- For a valuation $\alpha \in \mathbf{B}^n$ we call

  $$m(\alpha) = \bigwedge_{i=1}^{n} x_i^{\alpha_i} \quad (\text{Notation: } x_i^1 := x_i, x_i^0 := x_i')$$

  the **minterm associated with** α.
- A disjunction of pairwise different monomials is called polynomial.
  If all monomials in a polynomial are minterms, then the polynomial is complete.

Normal forms

- A **disjunctive normal form (DNF)** of a Boolean function f is a polynomial that describes f.
- A **canonical disjunctive normal form (CDNF)** of a Boolean function f is a complete polynomial that describes f.

Question: What do we mean by "canonical"?

Boolean functions/ Boolean expressions

Lemma:

For every Boolean function $f \in B_{n,1}$ there is a Boolean expression that describes f.

Proof:

We have that $f = \psi\left(\sum_{\alpha \in ON(f)} m(\alpha)\right)$

Remark:

There is *no unique* Boolean expression for a given Boolean function.
For every Boolean expression **h** we have $\psi(\mathbf{h}) = \psi(\mathbf{h}+\mathbf{h}) = \psi(\mathbf{h}+\mathbf{h}+\mathbf{h}) \ldots$

Canonical disjunctive normal form

$$f = \sum_{\alpha \in ON(f)} m(\alpha)$$

is called canonical disjunctive normal form (CDNF) of f.

- The CDNF of f is **unique**
  up to the order of the literals in the minterms and the order of the minterms in the polynomial.
- There are other "two-level" canonical normal forms, e.g., the canonical conjunctive normal form.

Central questions

1. Can every Boolean function be implemented by some circuit? ✓

2. Given a Boolean function, can we systematically construct a circuit that implements this function? ✓

3. Given a Boolean function, can we systematically construct an **efficient** circuit that implements this function? ?

Open questions

If there are many polynomials (Boolean expressions) for a given function f, how do we find a “cheap” one?

How can Boolean expressions (polynomials) be implemented in practice?

For the special case of polynomials:
programmable logic arrays (PLAs)