Logic and Discrete Structures -LDS

Course 9 – Propositional Logic

S.l. dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

What have we covered until now?

DISCRETE MATHEMATICS

Discrete Structures: Lists, Sets, Relations, Tuples, Graphs, Trees

FUNCTIONAL PROGRAMMING

In PYTHON

Logic - general notions

Propositional Logic

Syntax

Semantics

Binary decision diagrams (BDD)

Conjunctive normal form (CNF)

Let's start with something simple

If X1+ X2 = 10 and X1 - X2 = 4.

What is the value of X1?

X1 = 7

Let's continue with something more complicated

Does the equation $a^4 + b^4 + c^4 = d^4$ have positive integers as a solution?
Formulated by Euler in 1769

Solved only after 2 centuries:
a = 95.800
b = 217.519
c = 414.560
d = 422.481

Let's continue with something more complicated

Does equation 313*$(x^3 + y^3) = z^3$ have positive integer solutions?

After a few tries we would tend to say it doesn't, but the first solution of the equation has 100 digits.

Let's continue with something more complicated

Any even integer can be written as a sum of 2 prime numbers.
Is the sentence true or false?

Ex: 24 = 11 + 13

Nobody knows yet.

Logic is the basis of computer science

Logic circuits: described in Boolean algebra Digital logic, sem. 2

Computability: what can be computed algorithmically?

Formal methods: proving correctness of programs
Java sorting error (Timsort) corrected (2015)

Artificial intelligence: how do we represent and infer knowledge?

Testing and security: finding inputs and error paths, automated vulnerability exploitation

etc.

From the history of logic

Aristotle (4th century BC): first system of formal (rigorous) logic

Gottfried Wilhelm Leibniz (1646-1714): computational logic

logical reasoning can be reduced to mathematical calculation

George Boole (1815-1864): The Laws of Thought: modern logic, Boolean algebra (logic and sets)

Gottlob Frege (1848-1925): classical symbolic logic

Begriffsschift: formalisation of logic as the foundation of mathematics

Bertrand Russell (1872-1970): Principia Mathematica

(with A. N. Whitehead)

formalisation attempting to eliminate earlier paradoxes

Kurt Gödel (1906-1978): incompleteness theorems (1931): no consistent and complete axiomatization of arithmetic limitation of logic: either paradoxes or unprovable statements

Logica și calculatoarele

Logical demonstrations are reduced to calculations
(algorithms, programs)

A proof is a verification of a proposition by a set of logical deductions from a set of axioms.

Many problems in computer science
can be reduced to logic
and then solve automatically

We already know: the usual logical operators

Not ($\neg$), OR ($\vee$), AND ($\wedge$)

```
if (year % 4 == 0 and year % 100 == 0 or year % 400 == 0)
```

Truth tables:

p	$\neg p$
F	T
T	F

negation $\neg$ Not

p \ q	F	T
F	F	T
T	T	T

disjunction $\vee$ OR

p \ q	F	T
F	F	F
T	F	T

conjunction $\wedge$ AND

C: `!`
Python: `not`

C: `||`
Python: `or`

C: `&&`
Python: `and`

Logic - general notions

Propositional Logic

Syntax

Semantics

Binary decision diagrams

Conjunctive normal form

Propositional logic

Propositional logic is one of the simplest languages (language ⇒ we can express something) we can express problems by formulas in logic.

Discussion:
How we define a logical formula:
Its form (syntax) vs. its meaning (semantics).

How do we represent a formula? To operate effectively with it

What are proofs and logical reasoning?
how can we demonstrate? can anything be proved (or denied)?

How do we use logic to operate with other notions in computer science? (sets, relations, etc.)

Logical sentences

A (logical) sentence is a statement that is either true or false, but not both simultaneously.

Are they or are they not sentences?

$2 + 2 = 5$

$x + 2 = 4$

All prime numbers greater than 2 are odd.

$x^n + y^n = z^n$ has no integer nonzero solutions for any $n > 2$

If $x < 2$, then $x^2 < 4$

Logic allows us to reason precisely.

⇒ for this we must define it precisely

syntax (how it looks/is formed) and semantics (what it means)

Logic - general notions
Propositional Logic
Syntax
Semantics
Binary decision diagrams
Conjunctive normal form

Syntax of propositional logic

A language is defined by
its symbols
and the rules by which we correctly combine symbols (syntax)

Symbols of propositional logic:
sentences: usually denoted by the letters p, q, r , etc.
operators (logical connectors): negation ¬, implication → ,
parentheses ( )

Propositional logic formulas: defined by structural induction
(construct complex formulas from simpler ones)
A formula is:
any proposition (also called atomic formula)
$(\neg \alpha)$ if α is a formula
$(\alpha \rightarrow \beta)$ if α and β are formulas (α, β called subformulas)

Other logical operators (connectors)

We usually give minimal definitions (as few cases as possible) (any further reasoning must be done on all cases)

Known operators can be defined using ¬ and →:

$\alpha \wedge \beta \stackrel{def}{=} \neg(\alpha \rightarrow \neg\beta)$ (AND)

$\alpha \vee \beta \stackrel{def}{=} \neg\alpha \rightarrow \beta$ (OR)

$\alpha \leftrightarrow \beta \stackrel{def}{=} (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ (equivalence)

We skip redundant parentheses, defining operator precedence.

Order of precedence: ¬, ∧, ∨, →, ↔

The implication is associative to the right! $p \rightarrow q \rightarrow r = p \rightarrow (q \rightarrow r)$

The syntax does not define what a formula means. We will define the semantics later.

Syntax (concrete and abstract) vs. semantics

Syntax: a set of rules that defines the constructions of a language

(if something is not constructed correctly we cannot define its meaning)

Concrete syntax specifies the exact way of writing.

sentence ¬ formula formula ∧ formula formula ∨ formula

Abstract syntax: the structure of the formula in subformulae (sentence, negation of a formula, conjunction/disjunction of 2 formulas) is of interest, not the concrete symbols (∧, ∨), infix/prefix spelling,...

Logical implication →

p → q
p: in reasoning: hypothesis, premise
q: in reasoning: conclusion
Meaning: if p is true, then q is true (if-then)
if p is not true, we don't know anything about q (can be anyway)
So p → q is false only when p is true, but q is false

Truth table:

p	$p \rightarrow q$	q = F	q = T
F		T	T
T		F	T

Expressed with the usual connectors: $p \rightarrow q \;=\; \neg p \vee q$

Negation: $\neg(p \rightarrow q) = p \wedge \neg q$

Implication in everyday speech and logic

In natural language, "if ... then" often denotes causality
if it rains, I take the umbrella (because it rains)
In mathematical logic $\rightarrow$ does NOT mean causality
3 is odd $\rightarrow$ 2 is prime true implication, T $\rightarrow$ T
(but the fact that 2 is prime is not because 3 is odd)
In proofs, we use relevant (conclusion-related) assumptions

Speaking, we often say "if" thinking "if and only if"
(equivalence, a stronger notion!)
Example: If I exceed the speed limit, I get a penalty.
(but what if I don't?)

WARNING: false implies anything! (see truth table)
$\Rightarrow$ reasoning from a false premise can lead to any conclusion
$\Rightarrow$ a paradox ($A \wedge \neg A$) destroys confidence in a logical system

Implication: contrapositive, inverse, reciprocal

Given an implication $A \rightarrow B$, we define:

Converse: $B \rightarrow A$
Inverse: $\neg A \rightarrow \neg B$
Contrapositive: $\neg B \rightarrow \neg A$

The contrapositive is equivalent to the initial (direct) formula.
$A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A$

The inverse is equivalent to the converse.
$B \rightarrow A \Leftrightarrow \neg A \rightarrow \neg B$

$A \rightarrow B$ is NOT equivalent to $B \rightarrow A$ (reciprocal)

Logic - general notions
Propositional Logic
Syntax
Semantics
Binary decision diagrams
Conjunctive normal form

Semantica unei formule: funcții de adevăr

We define rigorously how we calculate the truth value of a formula = we give a semantics (meaning) to the formula (formula = syntactic notion)

A truth function v assigns to any formula

a truth value $\in \{T, F\}$ such that:

v (p) is defined for each atomic proposition p.

$$v(\neg a) = \begin{cases} T & \text{if } v(a) = F \quad F \\ & \text{if } v(a) = T \end{cases}$$

$$v(a \rightarrow \beta) = \begin{cases} F & \text{if } v(a) = T \text{ and } v(\beta) = F \\ T & \text{otherwise} \end{cases}$$

Exemple: $v((a \rightarrow b) \rightarrow c)$

for $v(a) = T$, $v(b) = F$, $v(c) = T$ we have

$v(a \rightarrow b) = F$ because $v(a) = T$ and $v(b) = F$

and $v((a \rightarrow b) \rightarrow c) = T$

Interpretations of a formula

An interpretation of a formula = an evaluation for its sentences

An interpretation satisfies a formula if it evaluates it to T.
We say that the interpretation is a model for that formula.

Example: for formula $a \wedge (\neg b \vee \neg c) \wedge (\neg a \vee c)$
the interpretation v (a) = T, v (b) = F, v (c) = T satisfies
the interpretation v (a) = T, v (b) = T, v (c) = T does not.

A formula can be:
tautology (valid): true in all interpretations
satisfiable: true in at least one interpretation
contradiction (not satisfiable): not true in any interpretation
contingency: true in some interpretations, false in others
(neither tautology nor contradiction)

Truth table

The truth table shows the truth value of a formula in all possible interpretations

2^n interpretations if the formula has n sentences

a	b	c	a →(b →c)
F	F	F	T
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

a	b	c	(a →b) →c
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

Two formulas are equivalent if they have the same truth table

Two formulas φ and ψ are equivalent if $\varphi \leftrightarrow \psi$ is a tautology

Boolean Algebra

On sets, ∪, ∩ and the complement form a Boolean algebra.

Also a Boolean algebra forms *in logic* ∧, ∨ and ¬ :

Commutativity : $A \vee B = B \vee A \quad A \wedge B = B \wedge A$

Associativity : $(A \vee B) \vee C = A \vee (B \vee C)$ și

$(A \wedge B) \wedge C = A \wedge (B \wedge C)$

Distributivity : $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ și

$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Identity : there are two values (here F and T) such that:

$A \vee \mathrm{F} = A \qquad A \wedge \mathrm{T} = A$

Complement: $A \vee \neg A = \mathrm{T} \quad A \wedge \neg A = \mathrm{F}$

Other properties (can be deduced from the above):

Idempotence : $A \wedge A = A \quad A \vee A = A$

Absorption : $A \vee (A \wedge B) = A \quad A \wedge (A \vee B) = A$

$\neg A \vee (A \wedge B) = \neg A \vee B$ *simplify the formula!*

Examples of tautologies

$a \vee \neg a$ $\qquad\qquad$ $\neg\neg a \leftrightarrow a$

de Morgan's Rules:

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

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

$$a \rightarrow (b \rightarrow c) \leftrightarrow (a \wedge b) \rightarrow c$$

$(p \rightarrow q) \wedge p \rightarrow q$ $\qquad\qquad$ $(p \rightarrow q) \wedge \neg q \rightarrow \neg p$

$p \wedge q \rightarrow p$ $\qquad\qquad$ $(p \vee q) \wedge \neg p \rightarrow q$

$$(p \rightarrow q) \rightarrow q$$

$$(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r)$$

Representation of Boolean formulas

It's good to have a representation:

- canonical (an object is represented in only one way) we have equality if and only if they have the same representation
- simple and compact (easy to implement / store)
- easy to process (simple / efficient algorithms)

Such a representation: binary decision diagrams (Bryant, 1986)

Logic - general notions
Propositional Logic
Syntax
Semantics
Binary decision diagrams
Conjunctive normal form

Decomposition by one variable

Fixing the value of a variable in a formula simplifies it.

Let $f = (a \vee b) \wedge (a \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$.

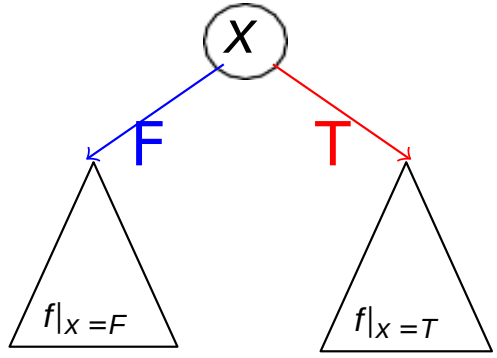
We give values to a: $f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c$

$f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$

Boolean (or Shannon) decomposition:

$$f = x \wedge f|_{x=T} \vee \neg x \wedge f|_{x=F}$$

expresses a Boolean function f
relative to a variable x

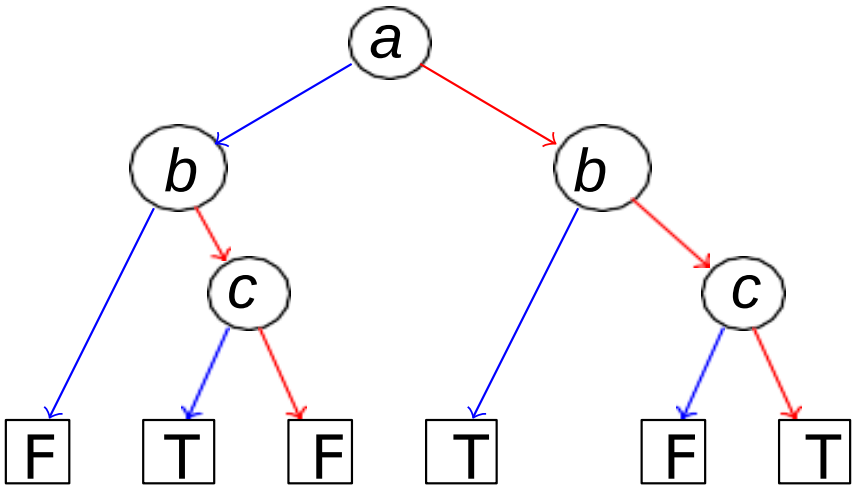


Binary decision tree

Continuing for the subformulas, we obtain a decision tree: giving values to the variables (a = T , b = F , c = T) and following the respective branches, we obtain the value of the function (T/F)

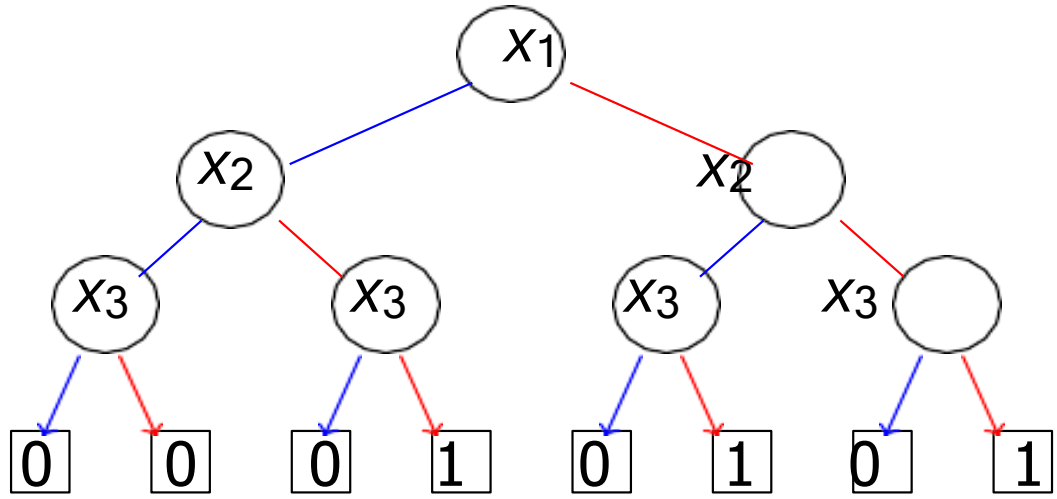
$f|_{a=T} = T \wedge T \wedge (\neg b \vee c) = \neg b \vee c$

$f|_{a=F} = b \wedge \neg c \wedge T = b \wedge \neg c$



Fixing the order of the variables, the tree is unique (canonical), but inefficient: 2^n possible combinations, like the truth table (though more compact)

From decision tree to binary decision diagram



$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$
ex: $f(T, F, T) = T$, $f(F, T, F) = F$, etc.

terminal nodes: function value (0 or 1, i.e. F or T)

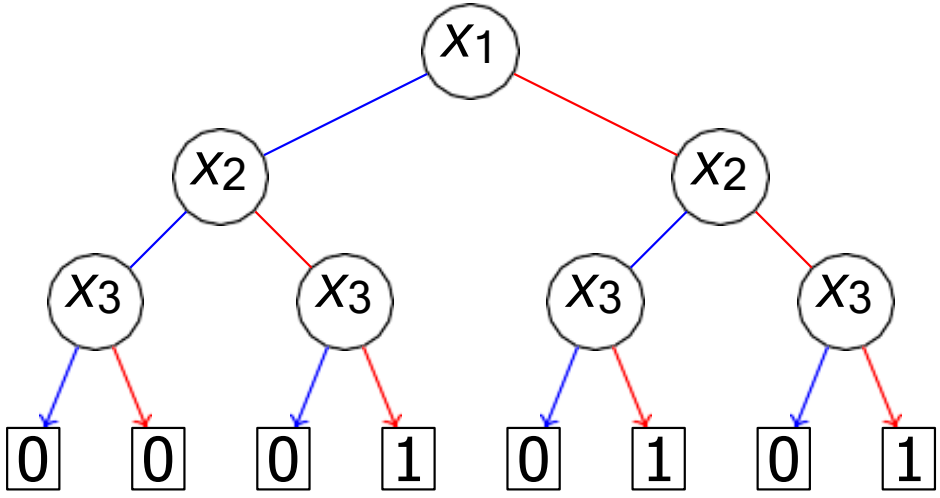
non-terminal nodes: variables xi (on which the function depends)

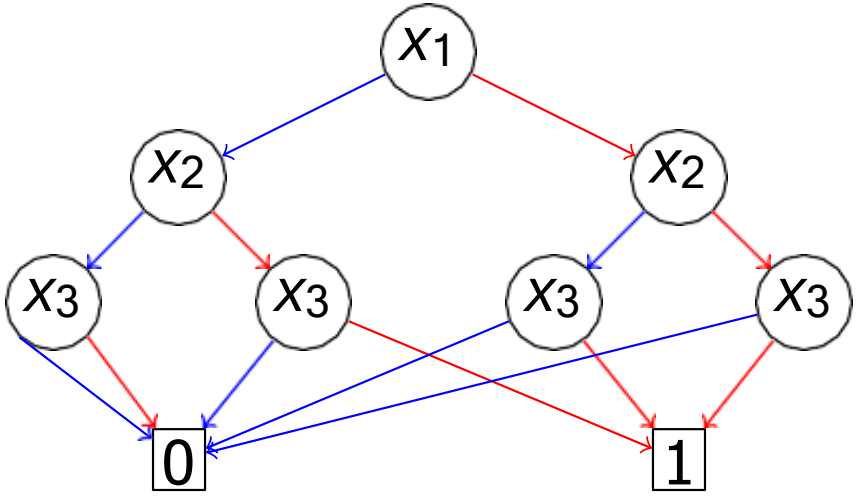
branches: low (node) / high(node) : F/T assignment of the variable in the node

We will define 3 transformation rules for a more compact form, binary decision diagram.

Reduction No. 1: Merging terminal nodes

We keep a single node for the values 0 and 1:

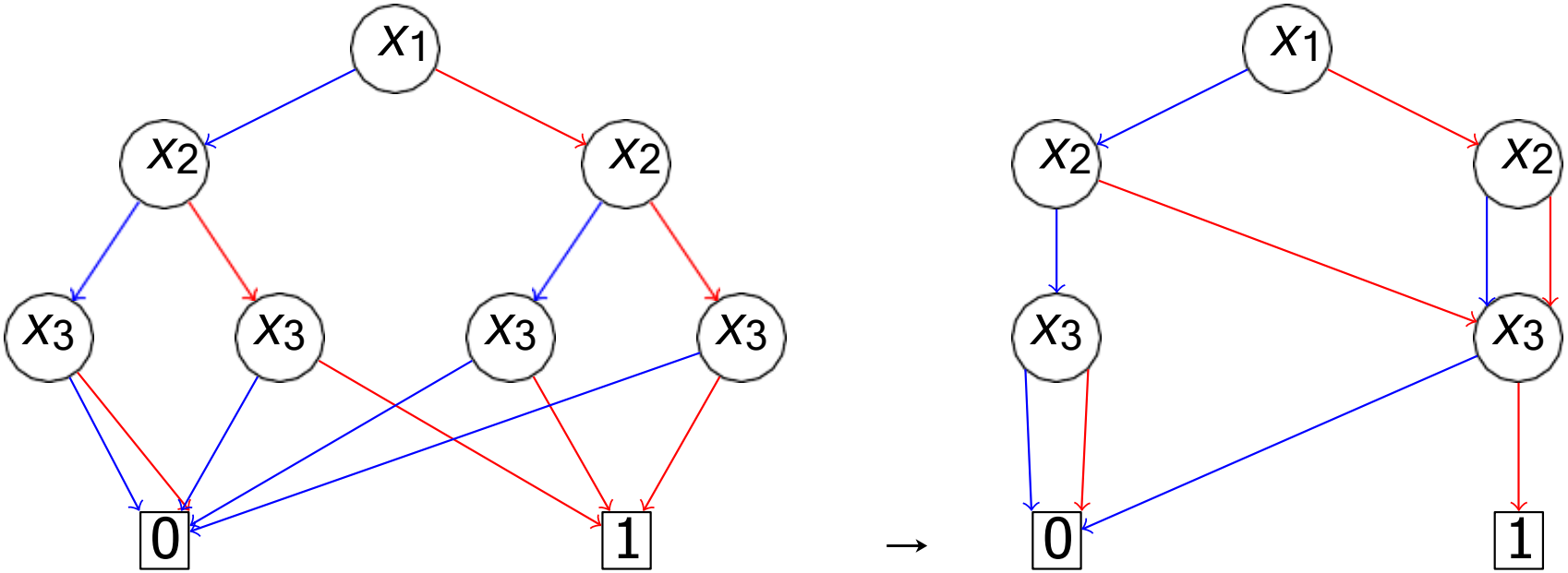




Reduction No. 2: Comasure of isomorphic nodes

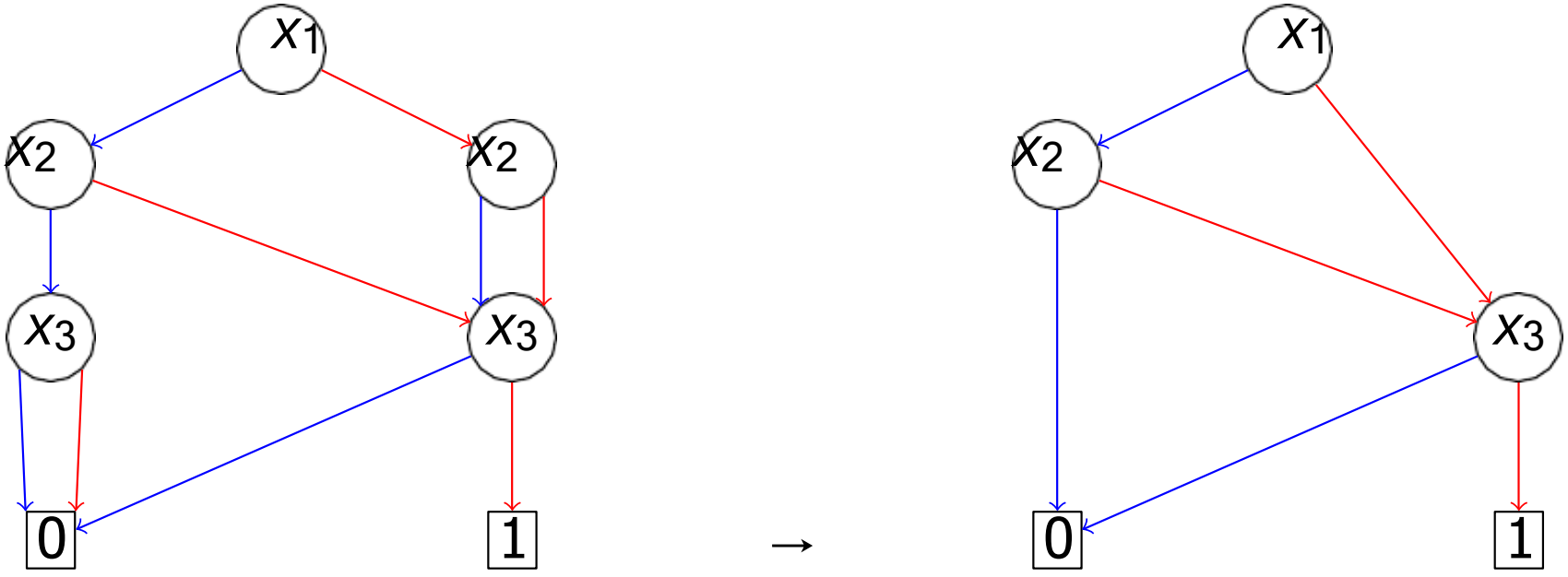
If low(n1)=low(n2) and high(n1)=high(n2), we merge n1 and n2

if they have the same result on the false branch and the same result on the true branch, the nodes give the same value:

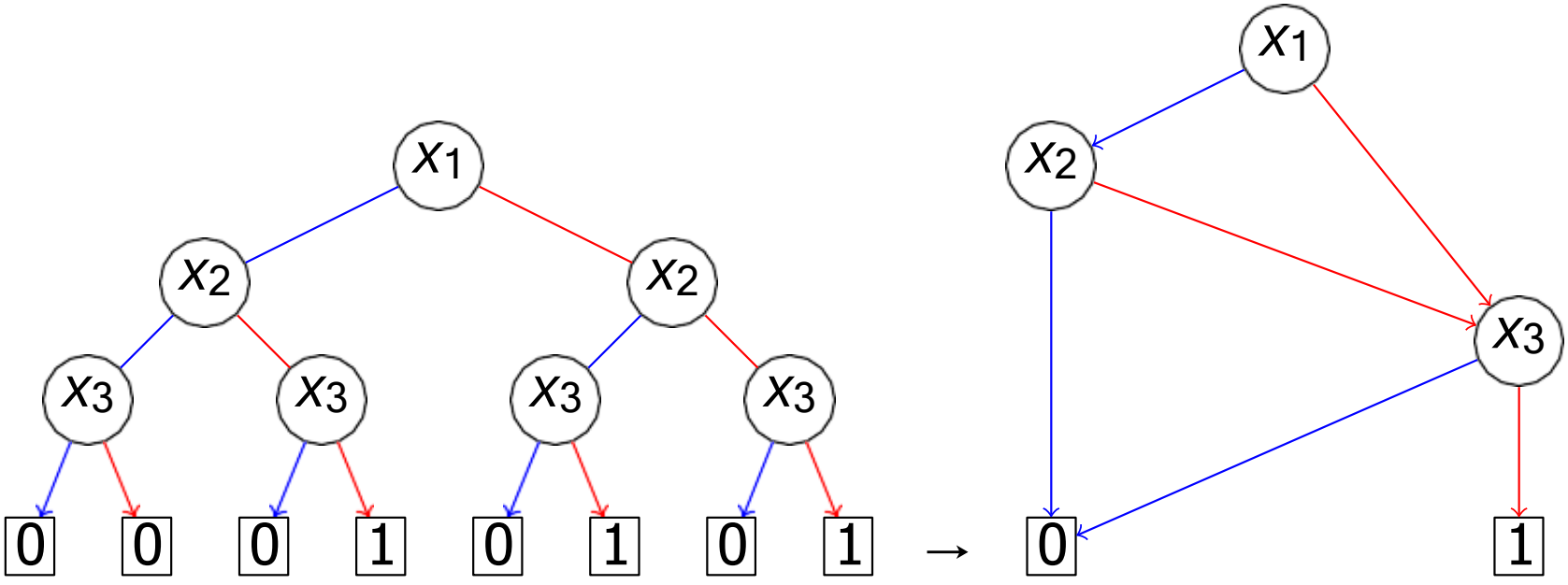


Reduction No 3: Eliminate unnecessary testing

Remove nodes with the same result on the false and true branches:



From tree to binary decision diagram



binary decision tree *binary decision diagram*

The three transformations are used to define a BDD.

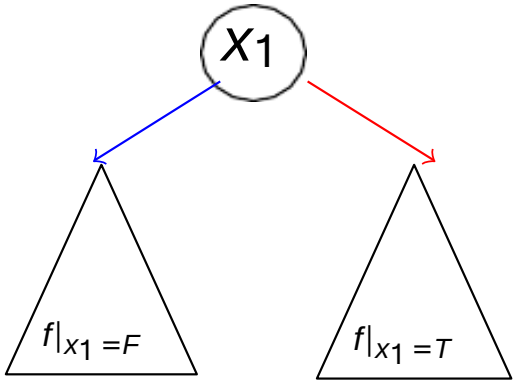
In practice, we want to avoid the decision tree as it is too large.

We directly apply function decomposition by a variable.

How to build a BDD in practice

In practice, we do NOT start from the complete binary tree.

We build a BDD directly recursively, decomposing after a variable:



$$f = x_1 \wedge f|_{x_1=T} \vee \neg x_1 \wedge f|_{x_1=F}$$

calculate $f|_{x_1=T}$ şi $f|_{x_1=F}$

then we merge the common nodes between the two parts

BDDs are used in almost all integrated circuit design software

To check the equality of two functions:
construct BDDs for the two functions if the functions are equal,

the same BDD is obtained
⇒ the equality of functions is directly and efficiently checked

Exemple: How to build a BDD

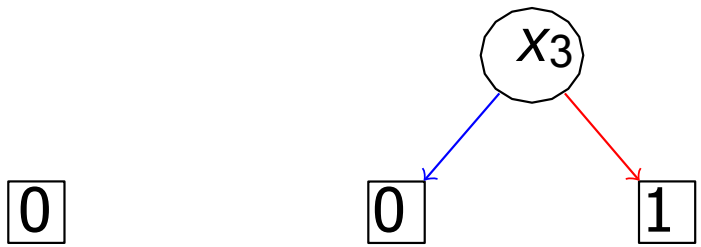
$f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$

We choose: x_1. Calculate $f|_{x_1=F}$ şi $f|_{x_1=T}$

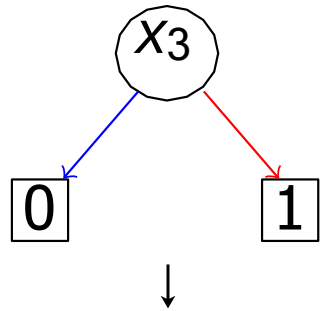
We build BDDs for the two functions: directly, if they are simple (T, F, p, $\neg p$),else we continue recursively,choosing a new variable:

$f_1 = f|_{x_1=F} = x_2 \wedge x_3$

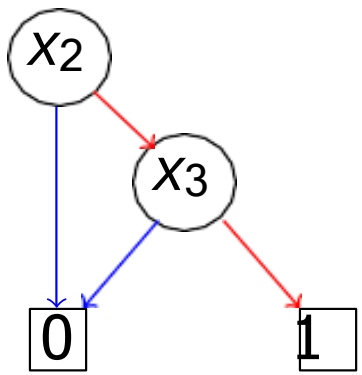
$f_1|_{x_2=F} = F$ $f_1|_{x_2=T} = x_3$



$f|_{x_1=T} = x_3$ f

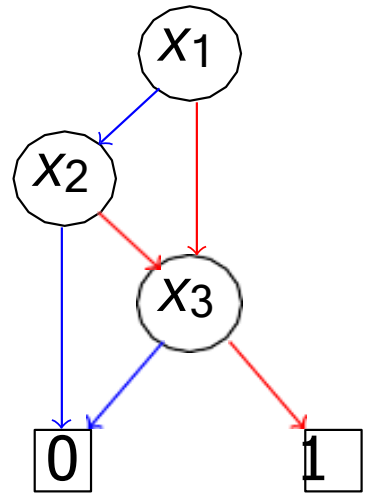


We add the decision node after x_2



We add the decision after x_1

→



We note that the diagram with x3 is common and we keep one copy

Logică - noțiuni generale
Logică Propozițională
Sintaxa
Semantica
Diagrame de decizie binară
Forma normală conjunctivă

Conjunctive normal form

used to determine whether a formula is feasible (can be T)

Def: *Conjunctive normal form*
= *conjunction* $\wedge$ of clauses
clause = *disjunction* $\vee$ of *literals*
literal = sentance or its negation
(p or $\neg p$)

$(a \vee \neg b \vee \neg d)$ clause
$\wedge\ (\neg a \vee \neg b)$ clause
$\wedge\ (\neg a \vee c \vee \neg d)$ ...
$\wedge\ (\neg a \vee b \vee c)$ clause

Similar: disjunctive normal form (disjunction of conjunctions)

Transformation to conjunctive normal form:

1) we carry (repeatedly) the negation inside - *de Morgan`s rules*

$$\neg(A \vee B) = \neg A \wedge \neg B \qquad \neg(A \wedge B) = \neg A \vee \neg B$$

2) we (repeatedly) carry the disjunction inside - *distributivity*

$$(A \wedge B) \vee C = (A \vee C) \wedge (B \vee C)$$

Example: conjunctive normal form

We work from the outside - avoiding unnecessary work

1) we take negations inside to sentences *de Morgan*

double negative disappears $\neg\neg A = A$

we replace the implications from the outside when we get to them

$p \to q = \neg p \vee q$ $\neg(p \to q) = p \wedge \neg q$

2) we take the disjunction $\vee$ inside the conjunction $\wedge$ *distributivity*

$\neg(\ (r \vee \neg(p \to (q \wedge r\ ))) \vee (p \wedge q)\ )$

$= \neg(r \vee \neg(p \to (q \wedge r\ ))) \wedge \neg(p \wedge q)$

$= \neg r \wedge (p \to (q \wedge r\ )) \wedge (\neg p \vee \neg q)$

$= \neg r \wedge (\neg p \vee (q \wedge r\ )) \wedge (\neg p \vee \neg q)$

$= \neg r \wedge (\neg p \vee q) \wedge (\neg p \vee r\ ) \wedge (\neg p \vee \neg q)$

Example 2: conjunctive normal form

$$
\begin{aligned}
& \neg((a \wedge b) \vee ((a \rightarrow (b \wedge c)) \rightarrow c)) \\
= \quad & \neg(a \wedge b) \wedge \neg((a \rightarrow (b \wedge c)) \rightarrow c)) \\
= \quad & (\neg a \vee \neg b) \wedge ((a \rightarrow (b \wedge c)) \wedge \neg c) \\
= \quad & (\neg a \vee \neg b) \wedge (\neg a \vee (b \wedge c)) \wedge \neg c \\
= \quad & (\neg a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee c) \wedge \neg c
\end{aligned}
$$

The transformation can exponentially increase the size of the formula:

$$
\begin{aligned}
& (a \wedge b \wedge c) \vee (p \wedge q \wedge r) \\
= \; & (a \vee (p \wedge q \wedge r)) \wedge (b \vee (p \wedge q \wedge r)) \wedge (c \vee (p \wedge q \wedge r)) \\
= \; & (a \vee p) \wedge (a \vee q) \wedge (a \vee r) \wedge (b \vee p) \wedge (b \vee q) \wedge (b \vee r) \\
& \wedge (c \vee p) \wedge (c \vee q) \wedge (c \vee r)
\end{aligned}
$$

In practice, auxiliary sentences are introduced ⇒ increases only linearly

Thank you!

Bibliografie

The content of the course is based on the material from the LSD course taught by Prof. Dr. Eng. Marius Minea and S.l. Dr. Eng. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)

The first example in the course (page 4) was taken from Stanford University's course CS221: Artificial Intelligence: Principles and Techniques (https://stanford.io/3ChWesU)

The examples in the first part of the course (pages 5-10) were taken from the course Mathematics for Computer Science at Massachusetts Institute of Technology (https://ocw.mit.edu/)