# LOGIC

- Logic involves interesting computation problems.
- Logic is "the calculus of computer science":
  digital circuit design, programming language semantics, specification and verification, constraint programming, logic programming, databases, artificial intelligence, knowledge representation, machine learning
- In computational complexity theory:
  computational problems from logic are of central importance; they can be used to express computation at various levels.
  This leads to important connections between complexity concepts and actual computational problems.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



# Boolean logic (propositional logic)

- Syntax (what are well-formed Boolean expressions):
  Boolean variables (atoms) $X = \{x_1, x_2, \ldots\}$
  Boolean connectives $\vee, \wedge, \neg$
- The set of Boolean expressions (formulae) is the smallest set such that all Boolean variables are Boolean expressions and if $\phi_1$ and $\phi_2$ are Boolean expressions, so are $\neg\phi_1$, $(\phi_1 \wedge \phi_2)$, and $(\phi_1 \vee \phi_2)$
  For example, $((x_1 \vee x_2) \wedge \neg x_3)$ is a Boolean expression but $((x_1 \vee x_2)\neg x_3)$ is not.
- An expression of the form $x_i$ or $\neg x_i$ is called a *literal* where $x_i$ is a Boolean variable.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Semantics (how to interpret Boolean expressions)

- Boolean expressions are propositions that are either true or false. They speak about a world where certain atomic proposition (Boolean variables) are either true or false.
  This induces are truth value for any Boolean expression as follows.
- A truth assignment $T$ is mapping from a finite subset $X' \subset X$ to the set of truth values $\{\mathbf{true}, \mathbf{false}\}$.
- Let $X(\phi)$ be the set of Boolean *variables appearing in* $\phi$.
  A truth assignment $T : X' \mapsto \{\mathbf{true}, \mathbf{false}\}$ is *appropriate* to $\phi$ if $X(\phi) \subseteq X'$.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Semantics—cont'd

- Let a truth assignment $T : X' \mapsto \{\mathbf{true}, \mathbf{false}\}$ be appropriate to $\phi$, i.e., $X(\phi) \subseteq X'$.
- $T \models \phi$ ($T$ *satisfies* $\phi$) is defined inductively as follows:
  If $\phi$ is a variable, then $T \models \phi$ iff $T(\phi) = \mathbf{true}$.
  If $\phi = \neg\phi_1$, then $T \models \phi$ iff $T \not\models \phi_1$
  If $\phi = \phi_1 \wedge \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ and $T \models \phi_2$
  If $\phi = \phi_1 \vee \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ or $T \models \phi_2$

**Example.** Let $T(x_1) = \mathbf{true}$, $T(x_2) = \mathbf{false}$.
Then $T \models x_1 \vee x_2$ but $T \not\models (x_1 \vee \neg x_2) \wedge (\neg x_1 \wedge x_2)$

Usual shorthands:

$\phi_1 \rightarrow \phi_2$: $\neg\phi_1 \vee \phi_2$ $\qquad$ $\phi_1 \leftrightarrow \phi_2$: $(\neg\phi_1 \vee \phi_2) \wedge (\neg\phi_2 \vee \phi_1)$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

## Logical equivalence

**Definition.** Expressions $\phi_1$ and $\phi_2$ are *equivalent* ($\phi_1 \equiv \phi_2$) iff for all truth assignments $T$ appropriate to both of them, $T \models \phi_1$ iff $T \models \phi_2$.

**Example.**

$(\phi_1 \vee \phi_2) \equiv (\phi_2 \vee \phi_1)$
$((\phi_1 \wedge \phi_2) \wedge \phi_3) \equiv (\phi_1 \wedge (\phi_2 \wedge \phi_3))$
$\neg\neg\phi \equiv \phi$
$((\phi_1 \wedge \phi_2) \vee \phi_3) \equiv ((\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3))$
$\neg(\phi_1 \wedge \phi_2) \equiv (\neg\phi_1 \vee \neg\phi_2)$
$(\phi_1 \vee \phi_1) \equiv \phi_1$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Some notation

- Simplified notation:

  $(((x_1 \vee \neg x_3) \vee x_2) \vee x_4 \vee (x_2 \vee x_5))$ is written as

  $x_1 \vee \neg x_3 \vee x_2 \vee x_4 \vee x_2 \vee x_5$

  or

  $x_1 \vee \neg x_3 \vee x_2 \vee x_4 \vee x_5$

- $\bigvee_{i=1}^{n} \varphi_i$ stands for $\varphi_1 \vee \cdots \vee \varphi_n$

  $\bigwedge_{i=1}^{n} \varphi_i$ stands for $\varphi_1 \wedge \cdots \wedge \varphi_n$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Normal forms

**Theorem.** Every Boolean expression is equivalent to one in conjunctive (disjunctive) normal form.

CNF: $(l_{11} \vee \cdots \vee l_{1n_1}) \wedge \cdots \wedge (l_{m1} \vee \cdots \vee l_{mn_m})$
DNF: $(l_{11} \wedge \cdots \wedge l_{1n_1}) \vee \cdots \vee (l_{m1} \wedge \cdots \wedge l_{mn_m})$
where each $l_{ij}$ is a literal (Boolean variable or its negation).

A disjunction $l_1 \vee \cdots \vee l_n$ is called a *clause*.

A conjunction $l_1 \wedge \cdots \wedge l_n$ is called an *implicant*.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Normal forms—cont'd

CNF/DNF transformation:

1. remove $\leftrightarrow$ and $\rightarrow$:

   $\alpha \leftrightarrow \beta \quad \leadsto \quad (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha) \quad (1)$

   $\alpha \rightarrow \beta \quad \leadsto \quad \neg\alpha \vee \beta \quad (2)$

2. Push negations in front of Boolean variables:

   $\neg\neg\alpha \quad \leadsto \quad \alpha \quad (3)$

   $\neg(\alpha \vee \beta) \quad \leadsto \quad \neg\alpha \wedge \neg\beta \quad (4)$

   $\neg(\alpha \wedge \beta) \quad \leadsto \quad \neg\alpha \vee \neg\beta \quad (5)$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

3. CNF: move $\wedge$ connectives outside $\vee$ connectives:

$$\alpha \vee (\beta \wedge \gamma) \quad \leadsto \quad (\alpha \vee \beta) \wedge (\alpha \vee \gamma) \quad (6)$$

$$(\alpha \wedge \beta) \vee \gamma \quad \leadsto \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma) \quad (7)$$

DNF: move $\vee$ connectives outside $\wedge$ connectives:

$$\alpha \wedge (\beta \vee \gamma) \quad \leadsto \quad (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \quad (8)$$

$$(\alpha \vee \beta) \wedge \gamma \quad \leadsto \quad (\alpha \wedge \gamma) \vee (\beta \wedge \gamma) \quad (9)$$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



**Example.** Transform $(A \vee B) \rightarrow (B \leftrightarrow C)$ to CNF.

$$(A \vee B) \rightarrow (B \leftrightarrow C) \quad (1,2)$$

$$\neg(A \vee B) \vee ((\neg B \vee C) \wedge (\neg C \vee B)) \quad (4)$$

$$(\neg A \wedge \neg B) \vee ((\neg B \vee C) \wedge (\neg C \vee B)) \quad (7)$$

$$(\neg A \vee ((\neg B \vee C) \wedge (\neg C \vee B))) \wedge (\neg B \vee ((\neg B \vee C) \wedge (\neg C \vee B))) \quad (6)$$

$$((\neg A \vee (\neg B \vee C)) \wedge (\neg A \vee (\neg C \vee B))) \wedge (\neg B \vee ((\neg B \vee C) \wedge (\neg C \vee B))) \quad (6)$$

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

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

- We can assume that normal forms do not have repeated clauses/implicants or repeated literals in clauses/implicants (for example $(\neg B \vee \neg B \vee C) \equiv (\neg B \vee C)$).
- Normal form can be exponentially bigger than the original expression in the worst case.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Satisfiability and validity

- A Boolean expression $\phi$ is *satisfiable* iff there is a truth assignment $T$ appropriate to it such that $T \models \phi$.
- $\phi$ is *valid* (or *tautology*) iff for every truth assignment $T$ appropriate to it, $T \models \phi$. If $\phi$ is valid, we write $\models \phi$.
- $\phi$ is unsatisfiable iff $\neg\phi$ is valid.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



**An important computational problem:**

- *SAT problem*: Given $\varphi$ in CNF, is $\varphi$ satisfiable?

  Example: $(x_1 \vee \neg x_2) \wedge \neg x_1$ is satisfiable

  $(x_1 \vee \neg x_2) \wedge \neg x_1 \wedge x_2$ is not.
- SAT can be solved in $O(n^2 2^n)$ time (e.g., truth table method).
- SAT $\in$ **NP**.

  Nondeterministic Turing machine for $\varphi \in$ SAT:
  **for** all variables $x$ in $\varphi$ **do**
  choose nondeterminically: $T(x) :=$ **true** or $T(x) :=$ **false**;
  **if** $T \models \varphi$ **then** return "yes" **else** return "no"
- It is open whether SAT $\in$ **P**.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

## Horn clauses

- An interesting special case of SAT concerns *Horn clauses*, i.e., clauses (disjunction of literals) with *at most one positive literal*.

  Example:

  $\neg x_1 \vee x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_3$, $x_2$ are Horn clauses but $\neg x_1 \vee x_2 \vee x_3$ is not.

  A Horn clause with a positive literal is called an *implication* and can be written as: $(x_1 \wedge x_3) \rightarrow x_2$

  (or $\rightarrow x_2$ when there are no negative literals.)

- HORNSAT problem:

  Given a conjunction of Horn clauses, is it satisfiable?

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



HORNSAT $\in \mathbf{P}$:

Algorithm $hornsat(S)$
/* Determines whether $S \in$ HORNSAT in polynomial time */
$T := \emptyset$ /* $T$ is the set of true atoms */
**repeat**
  **if** there is an implication $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \rightarrow y$ in $S$
    such that $\{x_1, \ldots, x_n\} \subseteq T$ but $y \notin T$ **then**
    $T := T \cup \{y\}$
**until** $T$ does not change
**if** for all purely negative clauses $\neg x_1 \vee \cdots \vee \neg x_n$ in $S$,
  there is some literal $\neg x_i$ such that $x_i \notin T$ **then**
  return $S$ is satisfiable
**else** return $S$ is not satisfiable

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



## Boolean functions and expressions

- An $n$-ary Boolean function is a mapping $\{\mathbf{true}, \mathbf{false}\}^n \mapsto \{\mathbf{true}, \mathbf{false}\}$.

  For example, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$ are possible binary Boolean functions and $\neg$ is a unary function.

- A Boolean expression $\phi$ can be seen as an $n$-ary Boolean function $f_\phi$ where $n = |X(\phi)|$.

- A Boolean expression $\phi$ with variables $x_1, \ldots, x_n$ *expresses* the $n$-ary function $f$ if for any $n$-tuple of truth values $\mathbf{t} = (t_1, \ldots, t_n)$, $f(\mathbf{t}) = \mathbf{true}$ if $T \models \phi$ and $f(\mathbf{t}) = \mathbf{false}$ if $T \not\models \phi$ where $T(x_i) = t_i$, $i = 1, \ldots, n$.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



**Proposition.** Any $n$-ary Boolean function $f$ can be expressed as a Boolean expression $\phi_f$ involving variables $x_1, \ldots, x_n$.

- The idea: model the rows of the truth table giving **true** as a disjunction of conjunctions.
- Let $F$ be the set of all $n$-tuples $\mathbf{t} = (t_1, \ldots, t_n)$ with $f(\mathbf{t}) = \mathbf{true}$.

  For each $\mathbf{t}$, let $D_\mathbf{t}$ be a conjunction of literals $x_i$ if $t_i = \mathbf{true}$ and $\neg x_i$ if $t_i = \mathbf{false}$.
- Let $\phi_f = \bigvee_{\mathbf{t} \in F} D_\mathbf{t}$

Note that $\phi_f$ is big in the worst case: $O(n2^n)$.
Not all Boolean functions can be expressed concisely.

**Example.**

| $x_1$ | $x_2$ | $f$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$\phi_f = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

## Boolean circuits

A more economical way to represent Boolean functions?

### Syntax

Graph $C = (V, E)$, $V = \{1, 2, \ldots, n\}$ (gates)
$C$ acyclic (for all $(i, j) \in E, i < j$).

All gates $i$ have a sort $s(i) \in \{\mathbf{true}, \mathbf{false}, \wedge, \vee, \neg\} \cup \{x_1, x_2, \ldots\}$.

if $s(i) \in \{\mathbf{true}, \mathbf{false}\} \cup \{x_1, x_2, \ldots\}$, indegree 0 (inputs)
if $s(i) = \neg$, indegree 1
if $s(i) \in \{\vee, \wedge\}$, indegree 2
Node $n$ output

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



### Semantics

Truth assignment $T : X(C) \mapsto \{\mathbf{true}, \mathbf{false}\}$ ($X(C)$: variables in $C$)

Truth value $T(i)$ for each gate $i$ inductively:

- If $s(i) = \mathbf{true}$, $T(i) = \mathbf{true}$ and if $s(i) = \mathbf{false}$, $T(i) = \mathbf{false}$.
- If $s(i) \in X(C)$, then $T(i) = T(s(i))$.
- If $s(i) = \neg$, then $T(i) = \mathbf{true}$ if $T(j) = \mathbf{false}$, otherwise $T(i) = \mathbf{false}$ where $(j, i)$ is the unique edge entering $i$.
- If $s(i) = \wedge$, then $T(i) = \mathbf{true}$ if $T(j) = T(j') = \mathbf{true}$ else $T(i) = \mathbf{false}$ where $(j, i)$ and $(j', i)$ are the two edges entering $i$.
- If $s(i) = \vee$, then $T(i) = \mathbf{true}$ if $T(j) = \mathbf{true}$ or $T(j') = \mathbf{true}$ else $T(i) = \mathbf{false}$ where $(j, i)$ and $(j', i)$ are the two edges to $i$.
- $T(C) = T(n)$ (*truth value of $C$*).

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



### Boolean circuits vs. Boolean expressions

- For each Boolean circuit $C$, there is a corresponding Boolean expression $\phi_C$.
- For each Boolean expression $\phi$, there is a corresponding Boolean circuit $C_\phi$ such that for any $T$ appropriate for both, $T(C_\phi) = \mathbf{true}$ iff $T \models \phi$.

  Idea: just introduce a new gate for each subexpression.
- Notice that Boolean circuits allow shared subexpressions but Boolean expressions do not.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio



### Important computational problems:

- CIRCUIT SAT:
  Given a circuit $C$, is there a truth assignment $T$ such that $T(C) = \mathbf{true}$?
- CIRCUIT SAT $\in \mathbf{NP}$
- CIRCUIT VALUE:
  Given a circuit $C$ with no variables, is it the case that $T(C) = \mathbf{true}$?
- CIRCUIT VALUE $\in \mathbf{P}$
  (No truth assignment is needed).

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

## Circuits compute Boolean functions

- A Boolean circuit with variables $x_1, \ldots, x_n$ *computes* an $n$-ary Boolean function $f$ if for any $n$-tuple of truth values $\mathbf{t} = (t_1, \ldots, t_n)$, $f(\mathbf{t}) = T(C)$ where $T(x_i) = t_i$, $i = 1, \ldots, n$.
- Any $n$-ary Boolean function $f$ can be computed by a Boolean circuit involving variables $x_1, \ldots, x_n$.
- Not every Boolean function can be computed using a concise circuit.

**Theorem.** For any $n \geq 2$ there is an $n$-ary Boolean function $f$ such that no Boolean circuit with $\frac{2^n}{2n}$ or fewer gates can compute it.

However, all natural families of Boolean functions seem to need only a linear number of gates to compute!

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio