**Thm:** The following problems are in polynomial time.

$$\begin{aligned}
\text{EmptyNFA} &= \{N \mid N \text{ is an NFA; } \mathcal{L}(N) = \emptyset\} \\
\Sigma^\star\text{DFA} &= \{D \mid D \text{ is a DFA; } \mathcal{L}(D) = \Sigma^\star\} \\
\text{MemberNFA} &= \{\langle N, w\rangle \mid N \text{ is an NFA; } w \in \mathcal{L}(N)\} \\
\text{EqualDFA} &= \{\langle D_1, D_2\rangle \mid D_1, D_2 \text{ DFAs; } \mathcal{L}(D_1) = \mathcal{L}(D_2)\} \\
\text{EmptyCFL} &= \{G \mid G \text{ is a CFG; } \mathcal{L}(G) = \emptyset\} \\
\text{MemberCFL} &= \{\langle G, w\rangle \mid G \text{ is a CFG; } w \in \mathcal{L}(G)\}
\end{aligned}$$

**Thm:** $\Sigma^\star$-CFL is co-RE complete.

We turn now to a unit on mathematical logic, the study of how mathematicians do mathematics. We model this process *as a piece of mathematics itself*, defining mathematical entities such as propositions and proof systems, and proving things *about them*.

Because our problems are so general and abstract, it is often hard to see exactly what real problems we are dealing with.

Logic is important to computer science in two main ways:

1. Because computers implement mathematically-defined rules, the results of logic tell us things about computability and complexity.
2. The problems of logic themselves provide applications for computing.

**Boolean variables:** $X = \{x_1, x_2, x_3, \ldots\}$

A boolean variable represents an atomic statement that may be either true or false.

**Boolean expressions:**

- atomic:: $x_i$, $\top$, $\bot$
- $(\alpha \vee \beta)$, $\neg\alpha$, for $\alpha, \beta$ Boolean exp's.

A *literal* is an atomic expression or its negation: $x_i$, $\neg x_i$, $\top$, $\bot$.

**Abbreviations:**

| | | |
|---|---|---|
| $\hookrightarrow$ | is an abbreviation for | "is an abbreviation for" |
| $(\alpha \wedge \beta)$ | $\hookrightarrow$ | $\neg(\neg\alpha \vee \neg\beta)$ |
| $(\alpha \rightarrow \beta)$ | $\hookrightarrow$ | $(\neg\alpha \vee \beta)$ |
| $(\alpha \leftrightarrow \beta)$ | $\hookrightarrow$ | $(\alpha \rightarrow \beta \ \wedge \ \beta \rightarrow \alpha)$ |

## Examples of boolean expressions:

- $x_1$
- $b_2 \vee \neg b_2$
- $x_1 \leftrightarrow x_2$
- $((a \rightarrow b) \wedge (b \rightarrow c)) \rightarrow (a \rightarrow c)$

**Truth assignment:** $T : X' \subseteq X \rightarrow \{\textbf{true}, \textbf{false}\}$

$$X(\varphi) \quad = \quad \{x_i \in X \mid x_i \text{ occurs in } \varphi\}$$

If $X(\varphi) \subseteq X'$, then $T$ is *appropriate* to $\varphi$. $T$ assigns truth value to $\varphi$:

$$\begin{array}{lcl} T \models \top & & T \not\models \bot \\ T \models x_i & \Leftrightarrow & T(x_i) = \textbf{true} \\ T \models (\alpha \vee \beta) & \Leftrightarrow & T \models \alpha \text{ or } T \models \beta \\ T \models \neg\alpha & \Leftrightarrow & T \not\models \alpha \end{array}$$

**Lemma 10.1**

$$\begin{array}{lcl} T \models (\alpha \wedge \beta) & \Leftrightarrow & T \models \alpha \textit{ and } T \models \beta \\ T \models \alpha \rightarrow \beta & \Leftrightarrow & T \not\models \alpha \textit{ or } T \models \beta \\ T \models \alpha \leftrightarrow \beta & \Leftrightarrow & T \models \alpha \textit{ iff } T \models \beta \end{array}$$

**Definition 10.2** $\alpha$ and $\beta$ are *semantically equivalent*

$$(\alpha \equiv \beta)$$

iff for all $T$ appropriate to $\alpha$ and $\beta$,

$$T \models (\alpha \leftrightarrow \beta)$$

♠

- $x_1 \ \equiv \ x_1 \vee \bot$
- $a \rightarrow a \ \equiv \ \top$
- $a \rightarrow b \ \equiv \ \neg b \rightarrow \neg a$
- $a \rightarrow b \ \equiv \ \neg a \vee b$
- $\neg(a \wedge b) \ \equiv \ \neg a \vee \neg b$
- $\neg(a \vee b) \ \equiv \ \neg a \wedge \neg b$
- $a \vee b \ \equiv \ b \vee a$
- $(a \vee b) \vee c \ \equiv \ a \vee (b \vee c)$
- $a \vee (b \wedge c) \ \equiv \ (a \vee b) \wedge (a \vee c)$
- $a \ \equiv \ \neg\neg a$

**Proposition 10.3** *Every boolean expression, $\varphi$, is equivalent to one in Conjunctive Normal Form (CNF), and to one in Disjunctive Normal Form (DNF).*

**Proof:** DNF: look at the truth table for $\varphi$:

| $x$ | $y$ | $z$ | $x \leftrightarrow y$ | $(x \leftrightarrow y) \leftrightarrow z$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$$(\bar{x} \wedge \bar{y} \wedge z) \ \vee \ (\bar{x} \wedge y \wedge \bar{z}) \ \vee \ (x \wedge \bar{y} \wedge \bar{z}) \ \vee \ (x \wedge y \wedge z)$$

CNF: put $\neg\varphi$ in DNF.

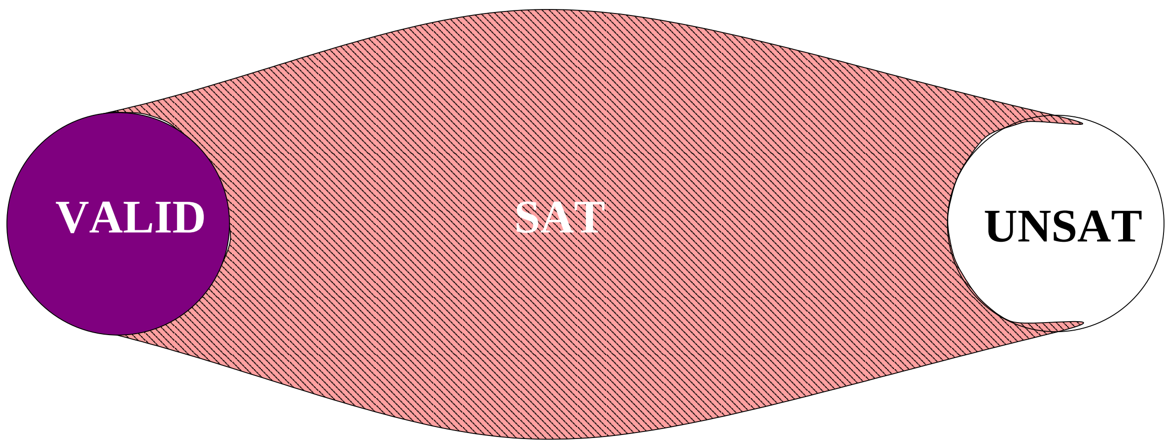
Use De Morgan's law:

$$\neg(C_1 \vee \cdots \vee C_k) \ \equiv \ (\neg C_1 \ \wedge \ \cdots \ \wedge \ \neg C_k)$$

♠

**Definition 10.4** A boolean expression $\varphi$ is *satisfiable* iff there exists $T \models \varphi$.

$\varphi$ is *valid* iff for all $T$ appropriate to $\varphi$, $T \models \varphi$. ♠




**Proposition 10.5** *For any boolean expression $\varphi$,*

$$\varphi \in \mathit{UNSAT} \qquad \Leftrightarrow \qquad \neg\varphi \in \mathit{VALID}$$

$$\text{UNSAT} \ \leq \ \text{VALID}; \qquad \text{VALID} \ \leq \ \text{UNSAT}$$

**Proposition 10.6** • $\varphi$ *is* unsatisfiable *iff* $\varphi \equiv \bot$.

- $\varphi$ *is* satisfiable *iff* $\varphi \not\equiv \bot$.
- $\varphi$ *is* valid *iff* $\varphi \equiv \top$.

**Proposition 10.7** SAT $\in$ **NP**

**Proof:** $\varphi \in$ SAT $\quad\Leftrightarrow\quad (\exists T)(T \models \varphi)$

Given $\varphi$, with $\quad X(\varphi) = \{x_1,\ x_2,\ x_3,\ \ldots,\ x_{n-1},\ x_n\ \}$

Nondeterminsitically, $\quad T := b_1,\ b_2,\ b_3,\ \ldots,\ b_{n-1},\ b_n$

**Accept** iff $T \models \varphi$

♠

Horn formulas are CNF formulas with at most one positive literal per clause. (Compare to PROLOG, not that I know anything about PROLOG.)

1. $(\bar{x} \vee y)$
2. $(\bar{x} \vee \bar{y} \vee \bar{z})$
3. $(x)$

1. $y \ \leftarrow \ x$
2. $\perp \ \leftarrow \ x, y, z$
3. $x \ \leftarrow \ \top$

**Theorem 10.8** HORN-SAT $\in$ **P**

**Algorithm 10.9** HORN-SAT$(\varphi)$

1. $T := \emptyset$ // no variables assigned true
2. **while** $(T \not\models \varphi)$ {
3. choose clause $\beta \leftarrow \alpha_1, \ldots, \alpha_r$ not satisfied
4. $T := T \cup \{\beta\}$ }
5. **if** $(\perp \in T)$ **then reject else accept**

2-SAT $=$

$$\{\varphi \in \text{SAT} \mid \varphi \in \text{CNF with two literals per clause}\}$$

$$\varphi_0 \;=\; (x_1 \vee \overline{x_2}) \;\wedge\; (x_2 \vee \overline{x_3}) \;\wedge\; (x_3 \vee x_1)$$

**Fact 10.10** 2-SAT $\in$ **P** *and in fact* 2-SAT *is complete for* **NSPACE**$[\log n]$.

Given a 2-CNF formula $\varphi$, define the directed graph $f(\varphi) = (V_\varphi, E_\varphi)$ as follows:

$$V_\varphi = \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$$

$$E_\varphi = \{\langle u, v\rangle \mid (\bar{u} \vee v) \text{ or } (v \vee \bar{u}) \text{ occurs in } \varphi.\}$$

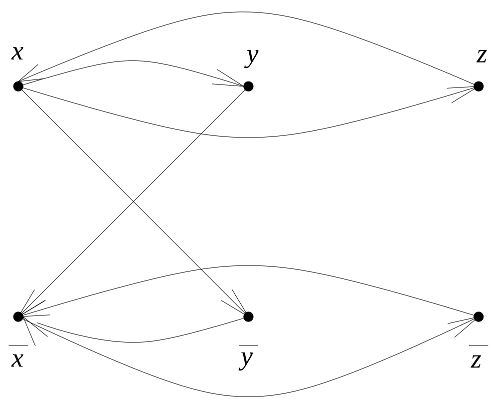
(Two bars cancel out, so $\overline{\overline{u}} = u$.)

$(\varphi \in \text{2-SAT}) \quad \Leftrightarrow \quad (\forall x \in X(\varphi))$"$x, \bar{x}$ not in same SCC"

SCC = strongly connected component

$(\varphi \in \text{2-SAT}) \quad \Leftrightarrow \quad (\forall x \in X(\varphi))$"$x, \bar{x}$ not in same SCC"

**Example:** $\varphi \;\equiv\; (\overline{x} \vee y) \wedge (\overline{y} \vee \overline{x}) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee x)$




There is a path from $x$ to $\overline{x}$, so $\overline{x}$ must hold.

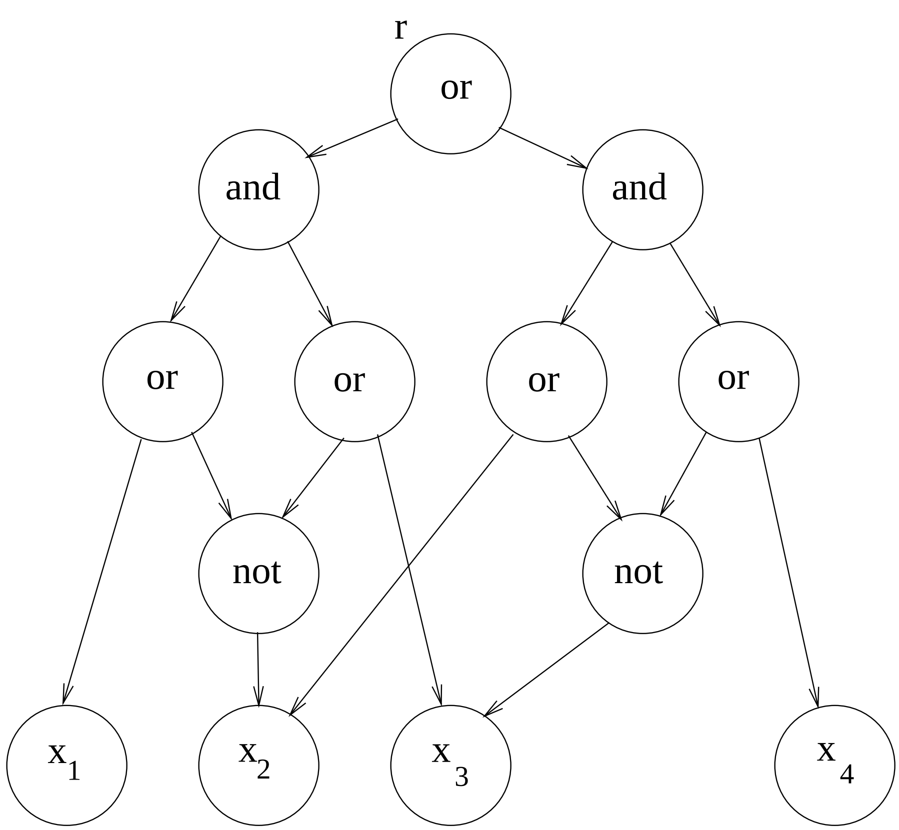
There is a path from $z$ to $\overline{z}$, so $\overline{z}$ must hold.

Either $y$ or $\overline{y}$ may hold; $\qquad \varphi \in \text{2-SAT}$

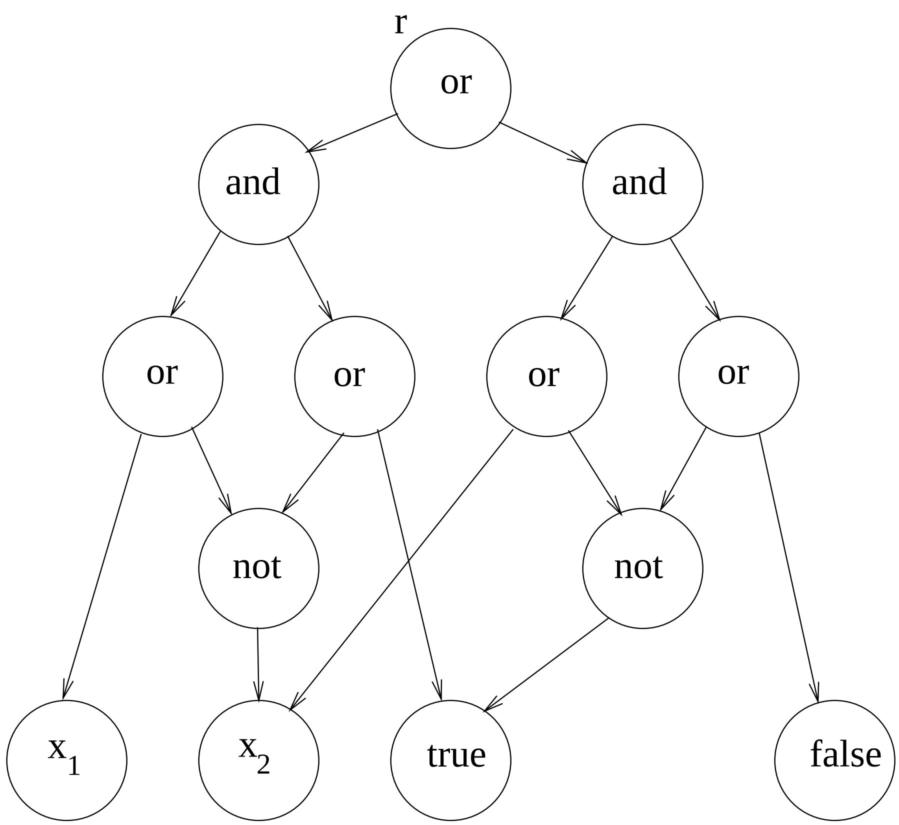
**Definition 10.11** A *boolean circuit* is a rooted directed, acyclic graph (DAG), $C = (V, E, s, r)$,

$$s : V \to \{\textbf{true}, \textbf{false}, \vee, \wedge \neg\} \ \cup \ \{x_1, x_2, \ldots\}$$

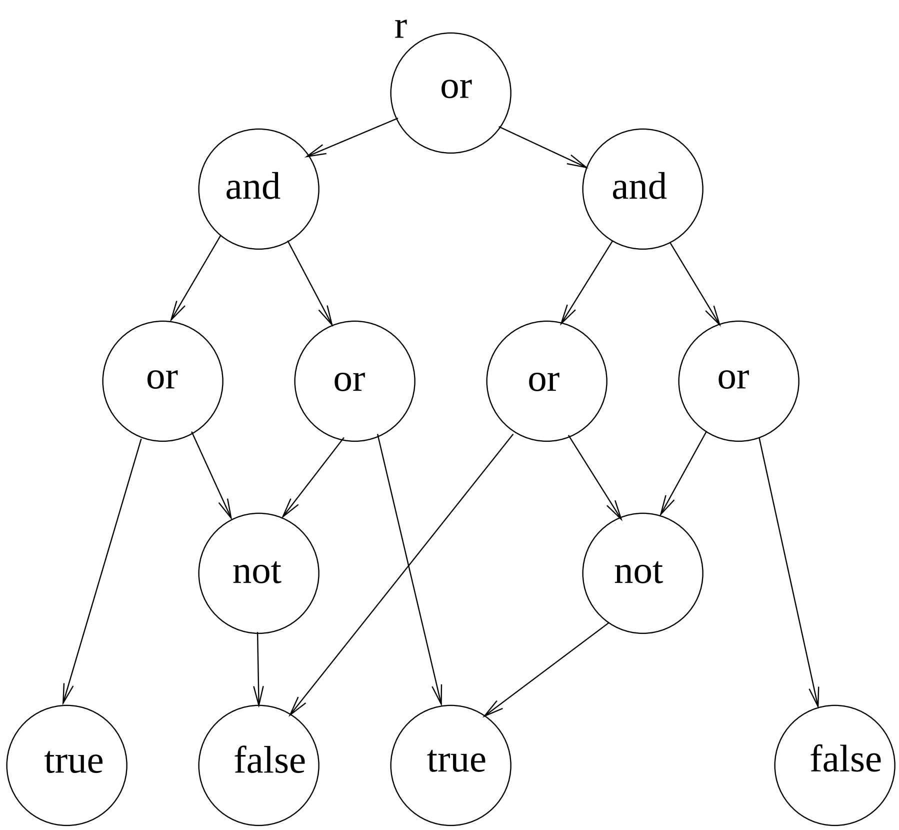
♠



**Proposition 10.12** Circuit-SAT $\in$ **NP**



**Proposition 10.13** *Circuit Value Problem (*CVP*)* $\in$ **P**



Circuits give a low-level model of computation, particularly of parallel computation (since gates on the same level operate in parallel).

$\mathcal{C} = \{C_1, C_2, C_3, \ldots\}$ a sequence of boolean circuits.
where $C_n$ has inputs $x_1, x_2, \ldots, x_n$

$$\mathcal{L}(\mathcal{C}) \quad = \quad \{w \in \{0,1\}^\star \mid C_{|w|}(w) = 1\}$$

Circuits are a hardware implementation of straight-line programs.

```
gate[1] = input[1]
gate[2] = input[2]
gate[3] = not gate[1]
gate[4] = not gate[2]
gate[5] = gate[1] and gate[4]
gate[6] = gate[2] and gate[3]
gate[7] = gate[5] or gate[6]
```

## Complexity Resources for Circuits:

- Size = number of gates and wires
- Depth = length of longest path from $r$ to leaf
- Uniformity = complexity of $f : n \mapsto C_n$

We define classes based on these, just as we defined classes based on time and space for Turing machines. We'll see much more about these classes later.