

# Recall From Last Time

**Prop:** The following functions are in **PrimRecFcns**:

$$M_1(n) = \mathbf{if} (n > 0) \mathbf{then} n - 1 \mathbf{else} 0$$

$$a \ominus b = \max(a - b, 0)$$

$$\mathbf{plus}(a, b) = a + b$$

$$\mathbf{mult}(a, b) = a \cdot b$$

$$\mathbf{exp}(a, b) = a^b$$

$$\mathbf{hyperexp}(a) = \mathbf{if} a = 0 \mathbf{then} 1 \mathbf{else} 2^{\mathbf{hyperexp}(a-1)}$$

$$==, \leq, <, >, \neq, P, L, R$$

**Prop: PrimRecFcns** is closed under

1. Definition by cases,
2. Bounded  $\sum$  and Bounded  $\prod$ , and
3. Bounded minimalization.

**Prop: PrimRecPreds** is closed under:

1. Boolean operations, and
2. Bounded quantification.

$$\text{Seq}(a_0, a_1, \dots, a_n) = 2^{a_0+1} 3^{a_1+1} \dots \text{PF}(n)^{a_n+1}$$

**Prop: IsSeq, length, Item**  $\in$  **PrimRecFcns**:

$$\text{IsSeq}(S) = \text{“}S \text{ is a Sequence number”}$$

$$\text{length}(\text{Seq}(a_0, a_1, \dots, a_n)) = n + 1$$

$$\text{Item}(\text{Seq}(a_0, a_1, \dots, a_n), i) = a_i$$

# Kleene's COMP Theorem

Let  $\text{COMP}(n, x, c, y)$  mean  $M_n(x) = y$ , and  $c$  is a sequence number encoding  $M_n$ 's entire computation on input  $x$ .  
Then  $\text{COMP} \in \mathbf{F}(\mathbf{Bloop})$ .

Encode TM computations:  $c = \text{Seq}(\text{ID}_0, \text{ID}_1, \dots, \text{ID}_t)$ .  
Each  $\text{ID}_j$  is a sequence number of tape-cell contents:

$$\text{ID}_j = \text{Seq}(\triangleright, a_1, \dots, a_{i-1}, [\sigma, a_i], a_{i+1}, \dots, a_r)$$

state:  $\sigma$ , head: cell  $i$ , tape:  $\triangleright, a_1, \dots, a_r, \sqcup, \sqcup, \dots$

$$\begin{aligned} \text{COMP}(n, x, c, y) \equiv & \\ & \text{START}(\text{Item}(c, 0), x) \wedge \text{END}(\text{Item}(c, \text{length}(c) - 1), y) \wedge \\ & \forall j < \text{length}(c) (\text{NEXT}(n, \text{Item}(c, j), \text{Item}(c, j + 1))) \end{aligned}$$

# Logic

Mathematical logic provides a mathematical model of the process of doing mathematics.

It is very abstract.

Its tools and methods are very useful and appropriate for computer science.

# Propositional Logic = Boolean Logic: Syntax

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

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

**Boolean expressions:** (also called **propositional formulas**)

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

**Literal:** atomic expression or its negation:  $x_i, \neg x_i, \top, \perp$ .

# Abbreviations

$\hookrightarrow$  is an abbreviation for “is an abbreviation for”

$$(\alpha \wedge \beta) \quad \hookrightarrow \quad \neg(\neg\alpha \vee \neg\beta)$$

$$(\alpha \rightarrow \beta) \quad \hookrightarrow \quad (\neg\alpha \vee \beta)$$

$$(\alpha \leftrightarrow \beta) \quad \hookrightarrow \quad (\alpha \rightarrow \beta \wedge \beta \rightarrow \alpha)$$

The fewer symbols we have, the fewer cases there will be in inductive proofs and definitions.

# Examples of Boolean Expressions

•  $x_1$

•  $b_2 \vee \neg b_2$

•  $x_1 \leftrightarrow x_2$

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

# Precedence

●  $\neg$

●  $\wedge, \vee$

●  $\rightarrow, \leftrightarrow$

$$\neg a \wedge b \rightarrow c \equiv ((\neg a) \wedge b) \rightarrow c$$

$$a \rightarrow b \rightarrow c \equiv a \rightarrow (b \rightarrow c)$$

# Boolean Logic: Semantics

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

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

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

$$T \models \top$$

$$T \not\models \perp$$

$$T \models x_i \quad \Leftrightarrow \quad T(x_i) = \mathbf{true}$$

$$T \models (\alpha \vee \beta) \quad \Leftrightarrow \quad T \models \alpha \text{ or } T \models \beta$$

$$T \models \neg\alpha \quad \Leftrightarrow \quad T \not\models \alpha$$

## Lemma:

$$T \models (\alpha \wedge \beta) \quad \Leftrightarrow \quad T \models \alpha \text{ and } T \models \beta$$

$$T \models \alpha \rightarrow \beta \quad \Leftrightarrow \quad T \not\models \alpha \text{ or } T \models \beta$$

$$T \models \alpha \leftrightarrow \beta \quad \Leftrightarrow \quad T \models \alpha \text{ iff } T \models \beta$$

## Proof:

Exercise: plug in definitions of abbreviations:  $\wedge, \rightarrow, \leftrightarrow$ . □

**Def:**  $\alpha$  and  $\beta$  are *semantically equivalent*,  $(\alpha \equiv \beta)$ ,  
iff for all  $T$  appropriate to  $\alpha$  and  $\beta$ ,  $T \models (\alpha \leftrightarrow \beta)$ ,

i.e.,  $\alpha$  and  $\beta$  have **identical truth tables**

$$x_1 \equiv x_1 \vee \perp$$

$$a \rightarrow a \equiv \top$$

$$a \rightarrow b \equiv \neg b \rightarrow \neg a$$

contrapositive

$$a \rightarrow b \equiv \neg a \vee b$$

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

de Morgan

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

de Morgan

$$a \vee b \equiv b \vee a$$

commutative

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

associative

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

distributive

$$a \equiv \neg\neg a$$

**Prop:** 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

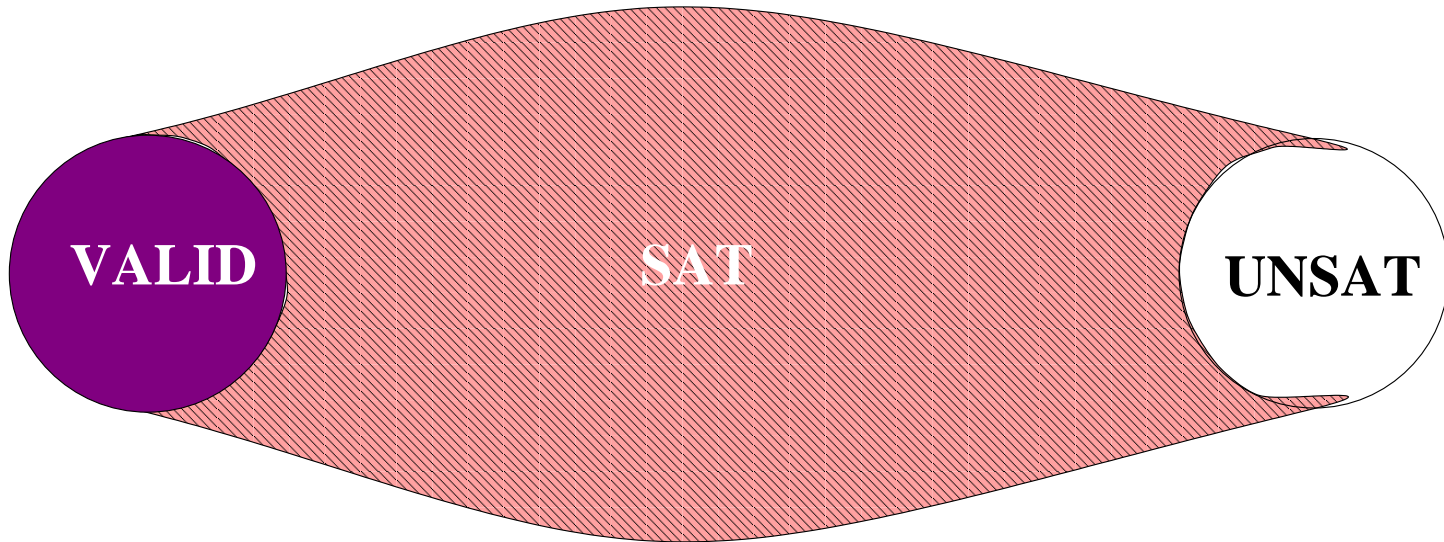
$$(\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge \bar{z}) \vee (x \wedge \bar{y} \wedge \bar{z}) \vee (x \wedge y \wedge z)$$

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

Use De Morgan's law:

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

**Def:** boolean expression  $\varphi$  is **satisfiable** iff exists  $T \models \varphi$ .  
 $\varphi$  is **valid** iff for all  $T$  appropriate to  $\varphi$ ,  $T \models \varphi$ .



**Prop:** For any boolean expression  $\varphi$ ,

$$\varphi \in \text{UNSAT} \quad \Leftrightarrow \quad \neg\varphi \in \text{VALID}$$

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

## Prop:

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

**Prop:**  $\text{SAT} \in \text{NP}$

**Proof:**  $\varphi \in \text{SAT} \iff \exists T (T \models \varphi)$

Given  $\varphi$ , with

$$\text{var}(\varphi) = \{x_1, x_2, x_3, \dots, x_{n-1}, x_n\}$$

Nondeterministically,

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

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



# Horn-SAT

Horn formulas are CNF formulas with at most one positive literal per clause.

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.  $x \leftarrow \top$

## Thm: HORN-SAT $\in$ P

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

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

# Thm: HORN-SAT $\in$ P

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

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

$y \leftarrow x$

$\perp \leftarrow x, y, z$       $T = \{ \quad \}$

$x \leftarrow \top$

# Thm: HORN-SAT $\in$ P

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

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

$y \leftarrow x$

$\perp \leftarrow x, y, z$       $T = \{ \quad x \quad \}$

$x \leftarrow \top$

# Thm: HORN-SAT $\in$ P

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

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

$y \leftarrow x$

$\perp \leftarrow x, y, z$

$T = \{ x, y \}$

$x \leftarrow \top$

**2-SAT** =  $\{\varphi \in \text{SAT} \mid \varphi \in \text{CNF}; \text{ 2 literals per clause}\}$

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

**Fact:** **2-SAT**  $\in$  **P**. In fact, **2-SAT** is complete for **NL**.

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

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

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

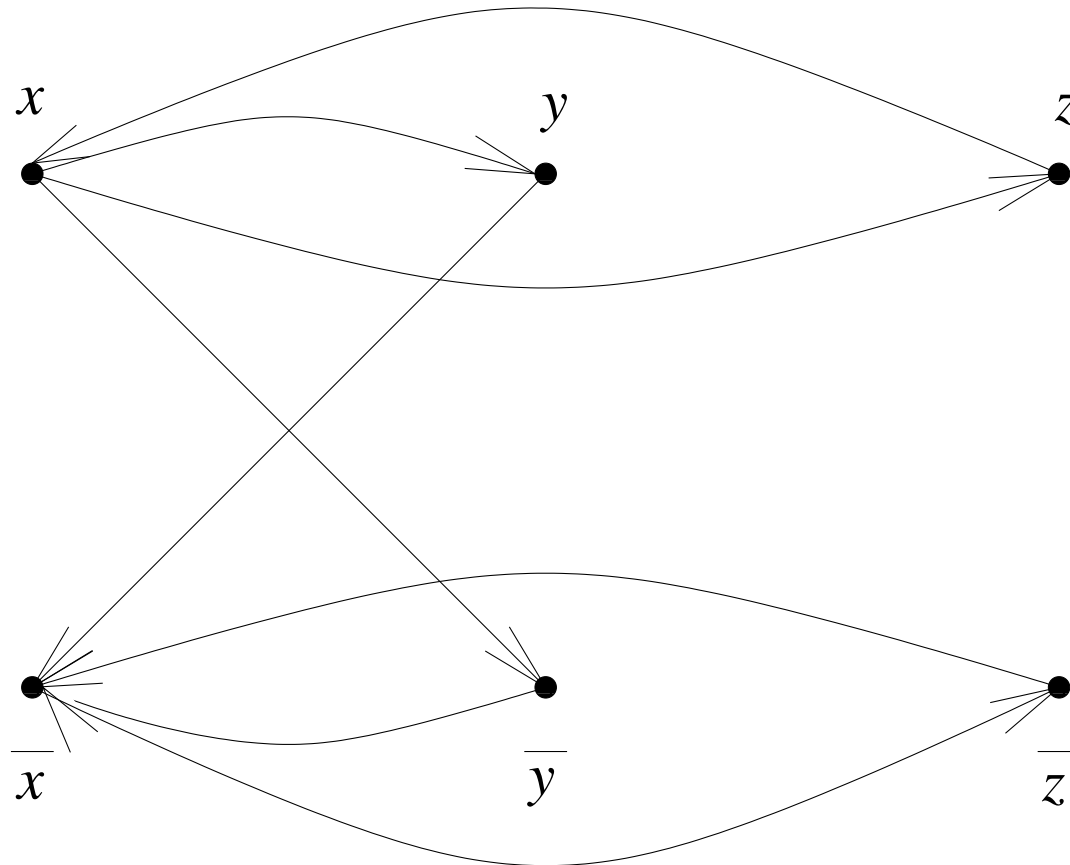
(Two bars cancel out, so  $\overline{\overline{u}} = u$ .)

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

**SCC** = strongly connected component

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

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

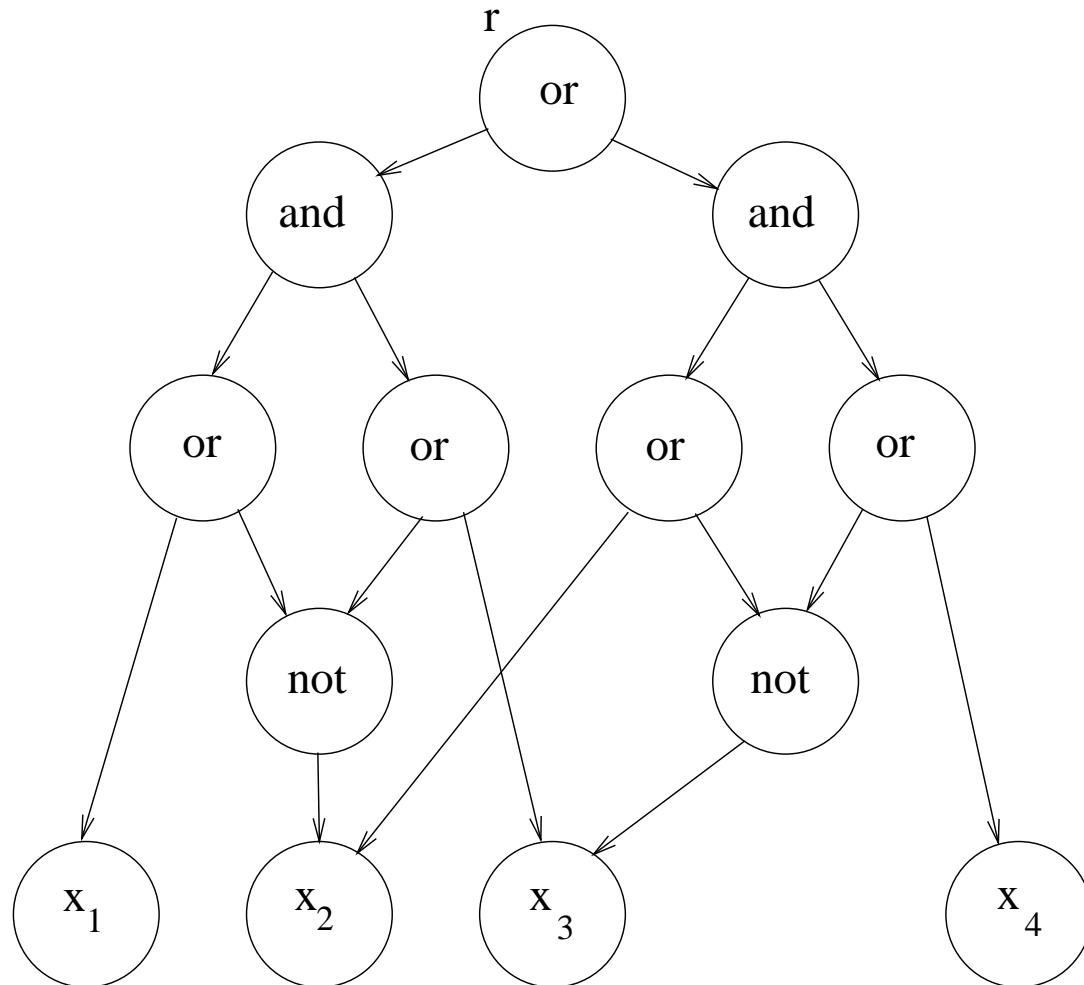


There is a path from  $x$  to  $\bar{x}$ , so  $\bar{x}$  must hold.

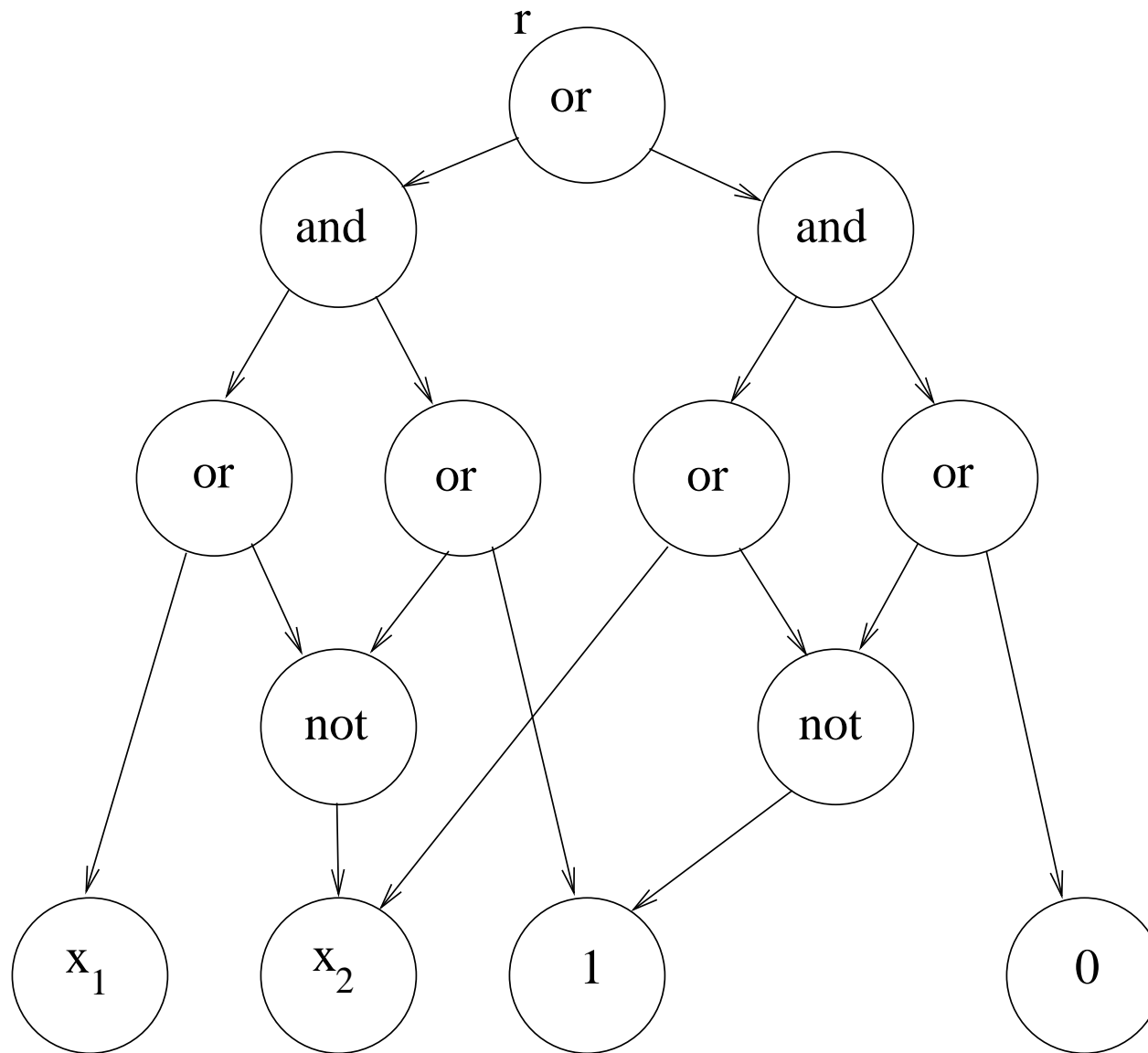
There is a path from  $z$  to  $\bar{z}$ , so  $\bar{z}$  must hold.

# Boolean Circuit: rooted, directed, acyclic graph

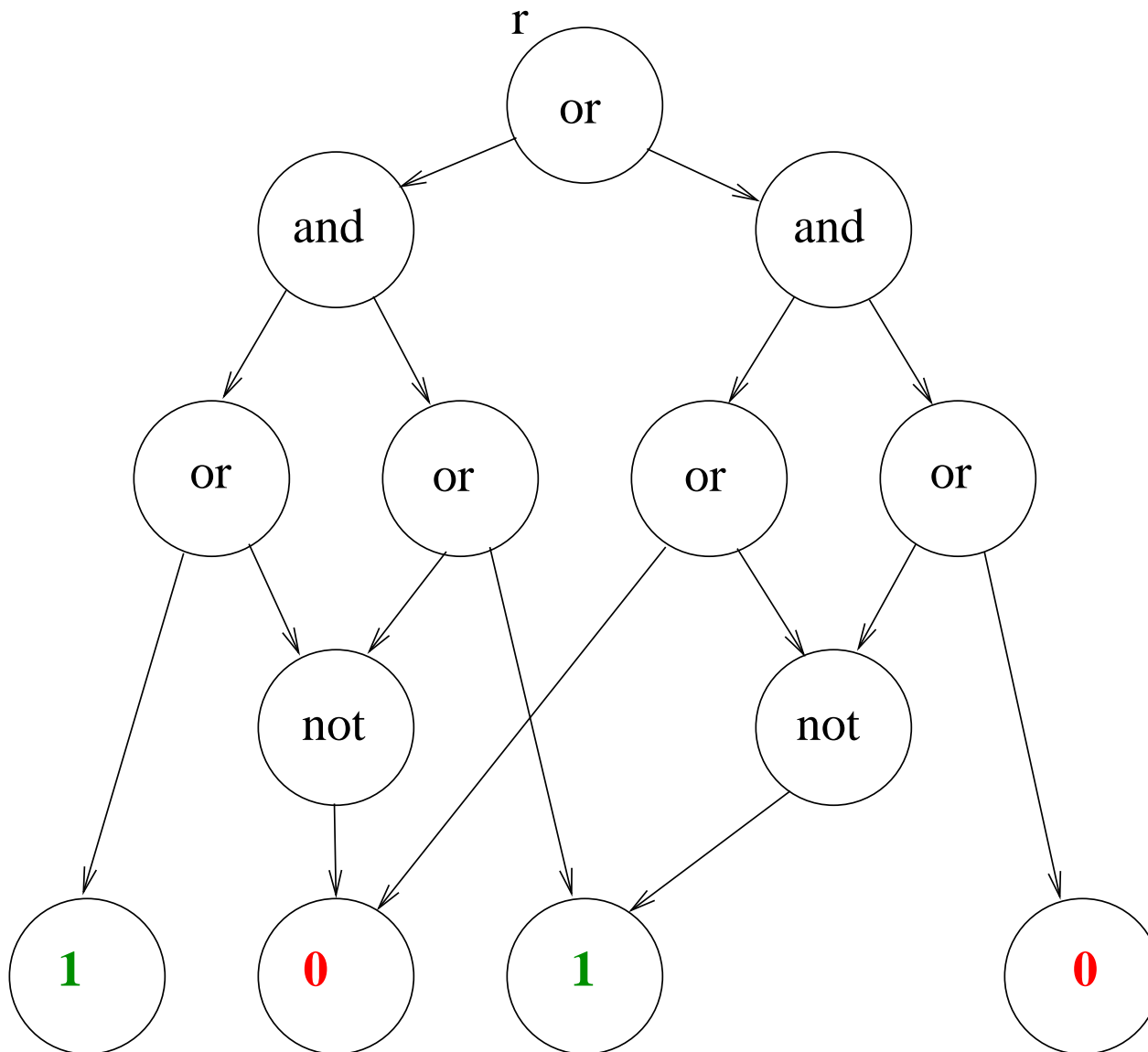
$$C = (V, E, s, r), \quad s : V \rightarrow \{0, 1, \vee, \wedge, \neg\} \cup \{x_1, x_2, \dots\}$$



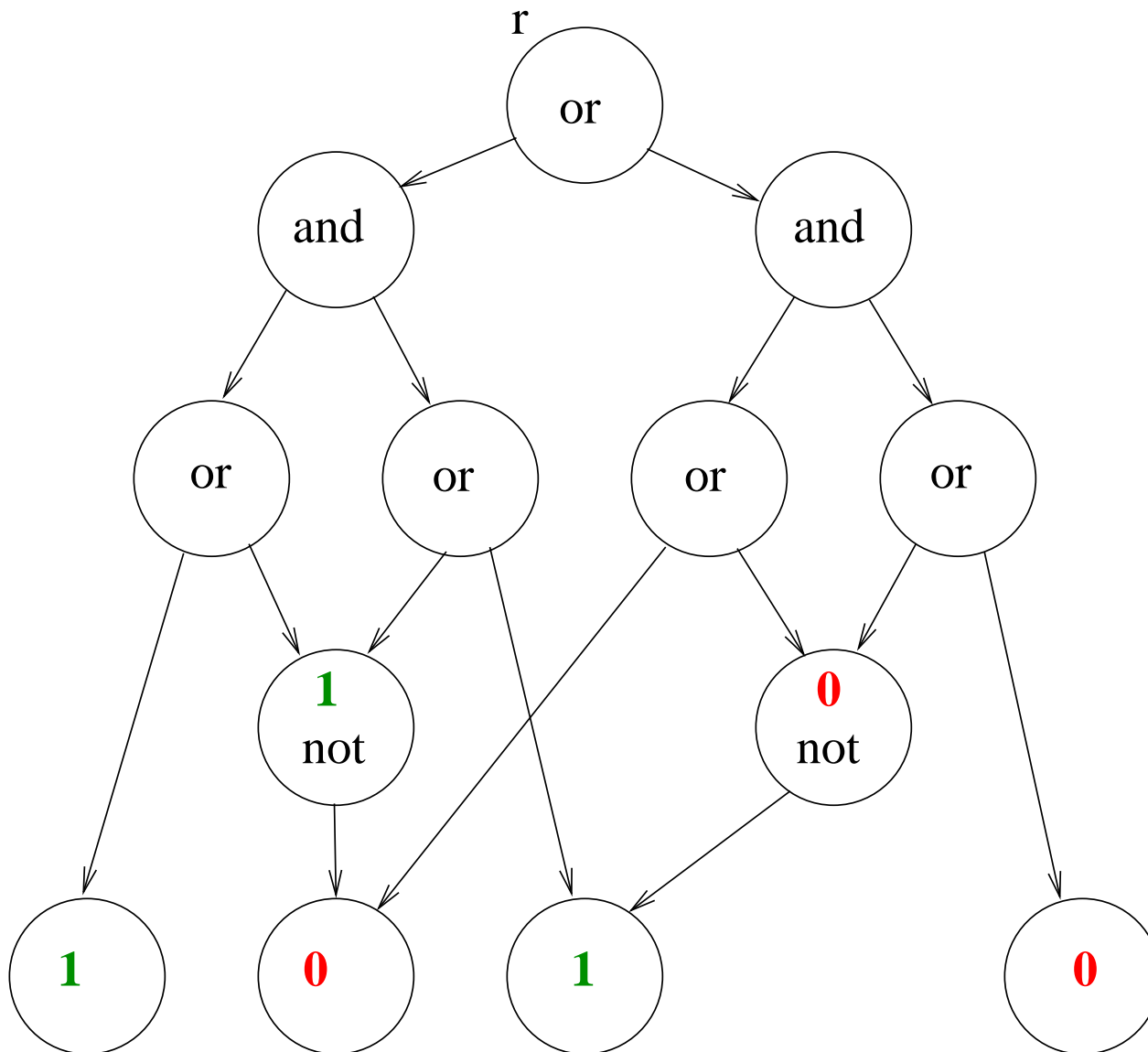
# Prop: Circuit-SAT $\in$ NP



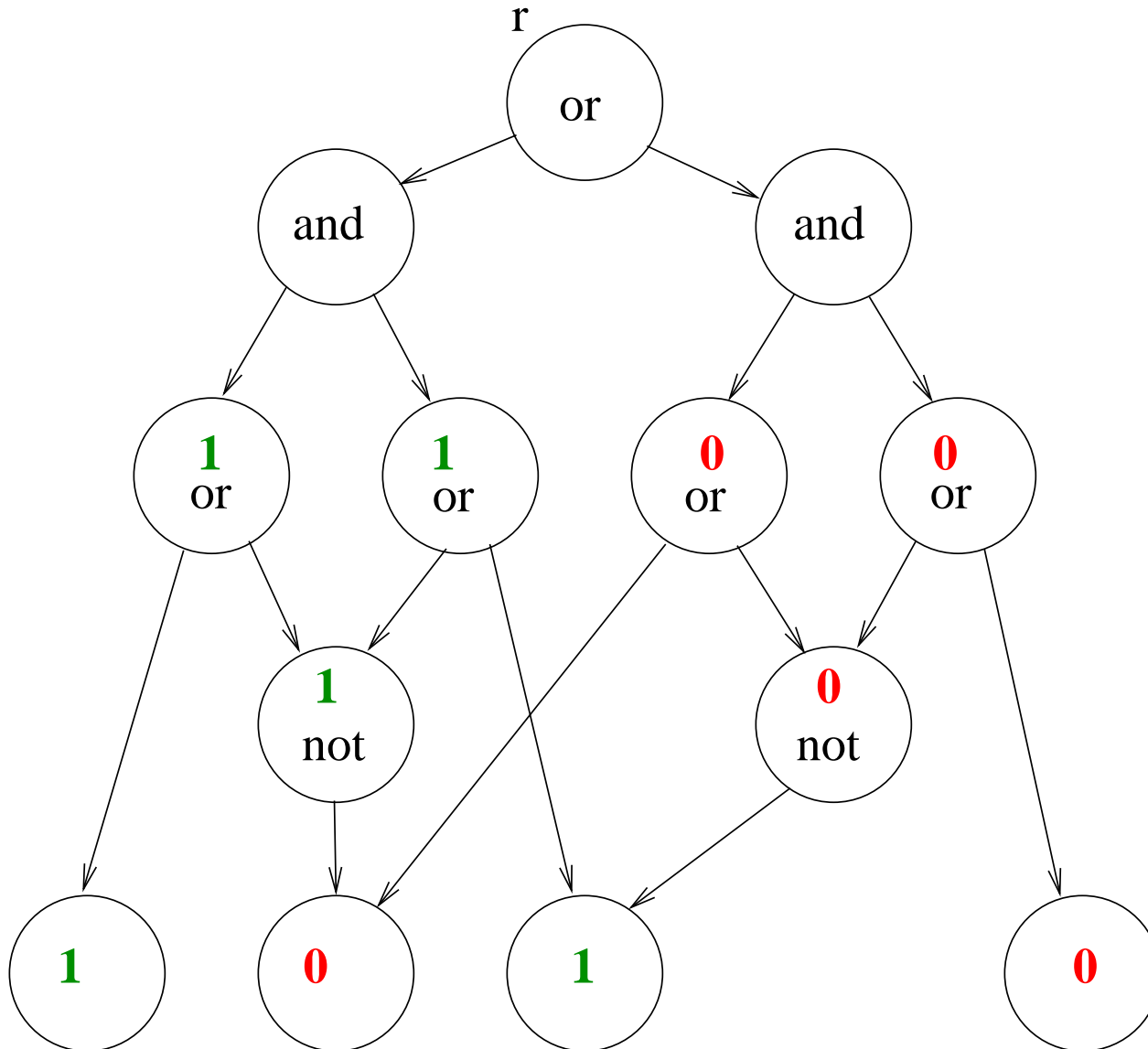
# Circuit Value Problem (CVP) $\in$ P



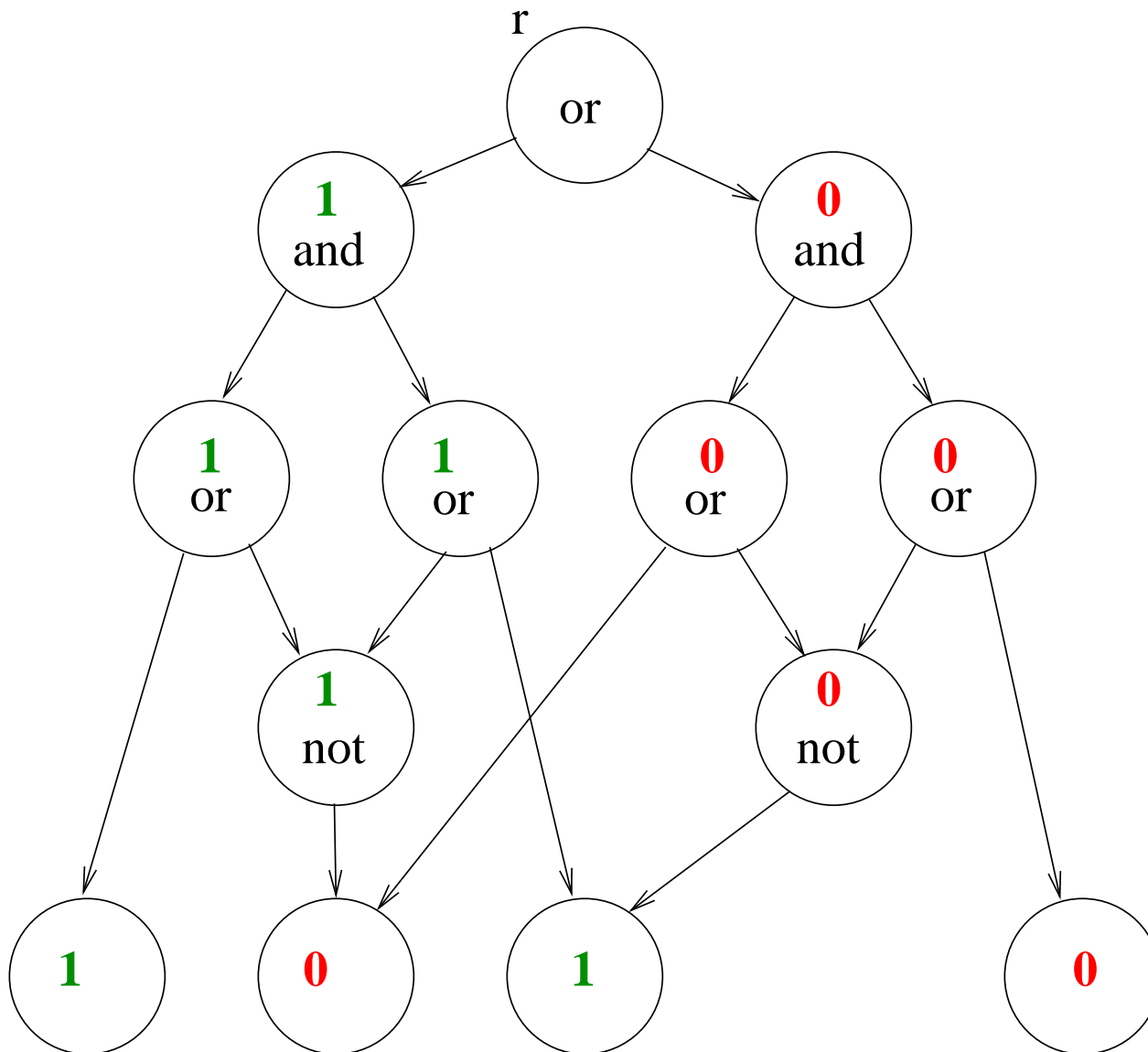
# Circuit Value Problem (CVP) $\in$ P



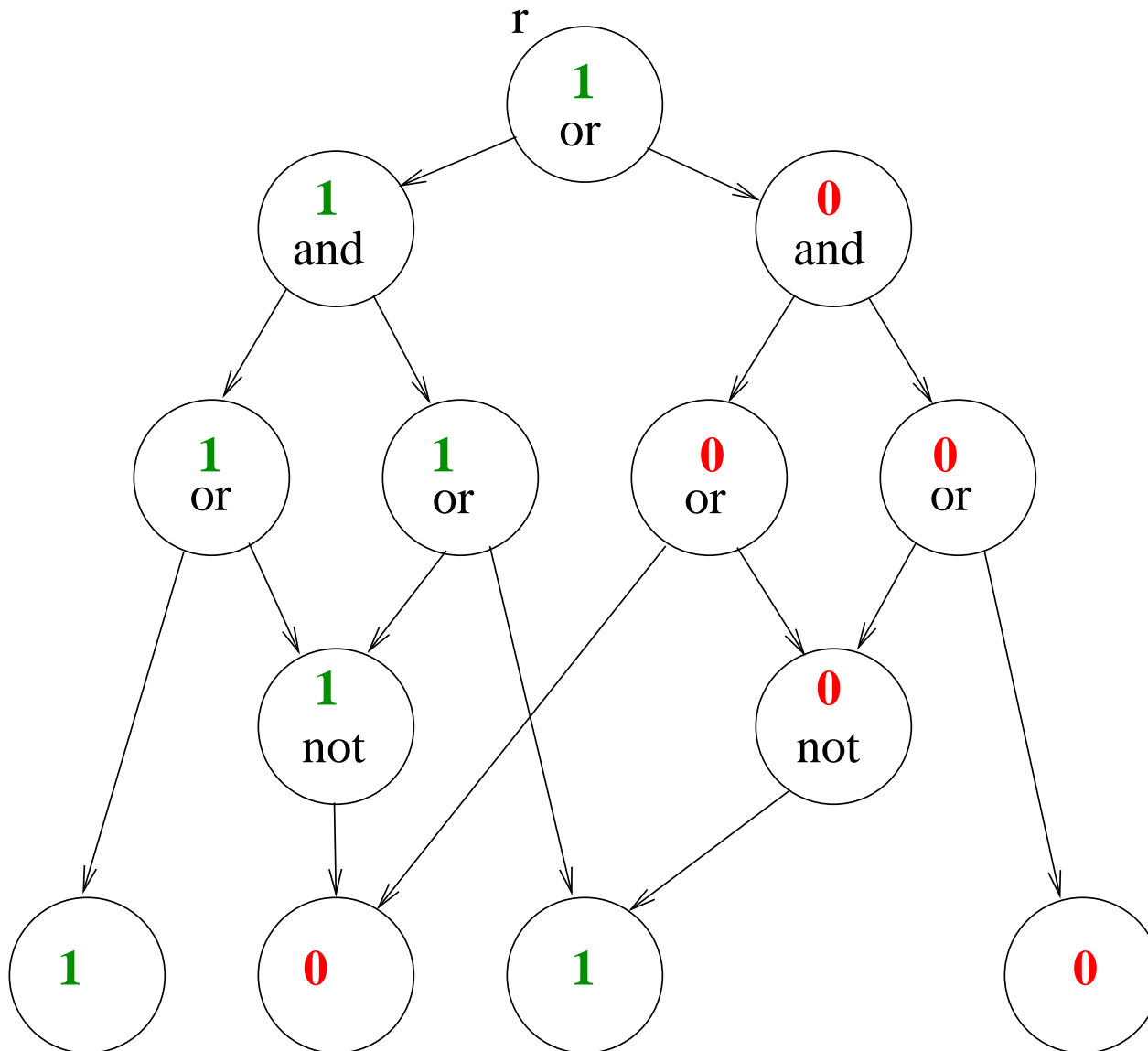
# Circuit Value Problem (CVP) $\in P$



# Circuit Value Problem (CVP) $\in$ P



# Circuit Value Problem (CVP) $\in$ P



# Circuit Classes

Circuits give a low-level model of computation.

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

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

Circuits are straight-line programs in hardware.

## Complexity Resources for Circuits:

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