More Models of Computation

Lecture 24

Context Free Grammars
Circuits
Decision Trees
Branching Programs

Context-Free Grammar

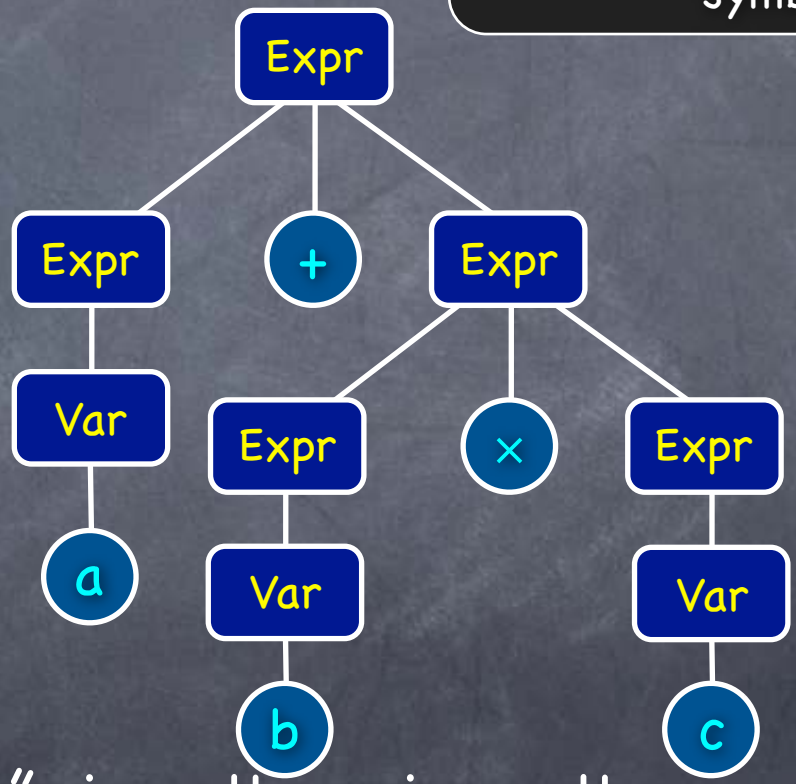
- Example: a (simplistic) syntax for arithmetic expressions

- Expr → Expr + Expr
  Expr → Expr × Expr
  Expr → Var
  Var → a
  Var → b
  Var → c

Set of "rewriting" rules over symbols

Also part of the grammar

- Start Symbol: Expr
  Terminals: +,×,a,b,c
- e.g. a + b × c



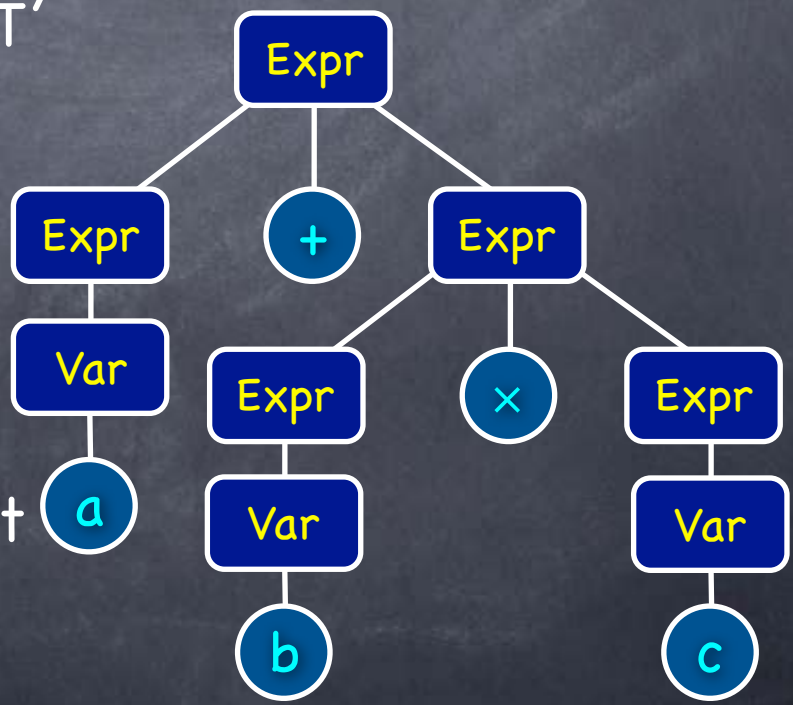
- (This grammar is "ambiguous" since there is another parse tree for the same string)

Context-Free Grammar

- Language of G: all strings "generated" by it
  - Defined in terms of all "parse trees" generated by it
- G generates the tree with one node, labeled with the start symbol
- If G generates T which has a leaf labeled with a non-terminal X, it also generates T' where the leaf is "expanded" using a rule $X \rightarrow \alpha_1 \ldots \alpha_t$
  - "Left-to-right order" of the children important while expanding
- Order in which expansions are done is not important (we only care about the tree)

Start Symbol: Expr
Terminals: +,×,a,b,c

Expr → Expr + Expr
Expr → Expr × Expr
Expr → Var
Var → a
Var → b
Var → c



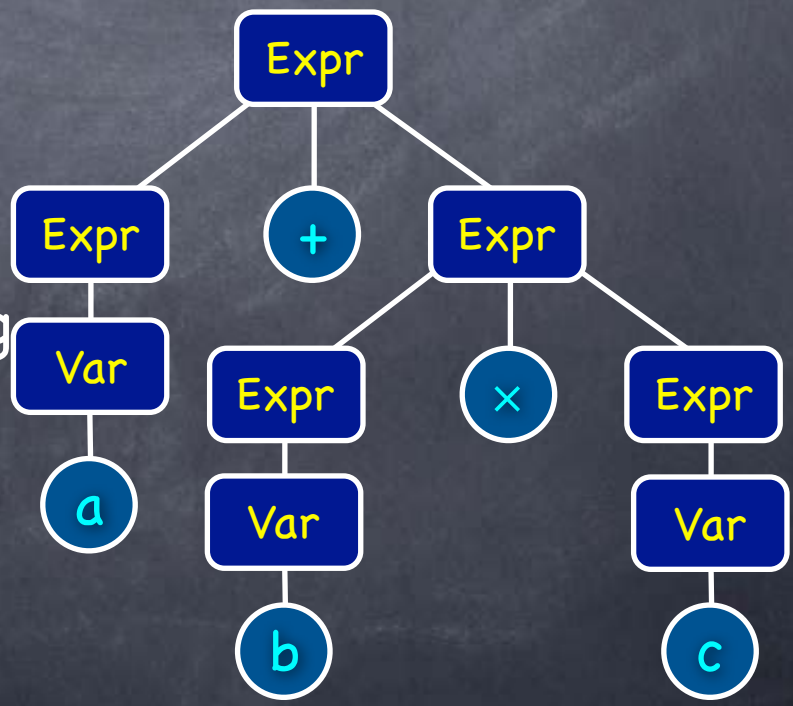
Context-Free Grammar

- Language of G: all strings "generated" by it
  - Defined in terms of all "parse trees" generated by it
- If G generates a tree T, with all leaves labeled by terminals, then G is said to generate the string of terminals obtained by reading the leaves left-to-right
  - Terminal ϵ denotes the empty string
  - e.g., S $\rightarrow$ SS | a | b | ϵ

    The string ab can be parsed as having no ϵ, or as (ϵa)(ϵb) or (ϵ)((ab)($\epsilon\epsilon$)) etc.
- If same string can be generated by different trees, an "ambiguous" grammar

Start Symbol: Expr
Terminals: +,×,a,b,c

Expr → Expr + Expr
Expr → Expr × Expr
Expr → Var
Var → a
Var → b
Var → c



NNXS

Question

- Which of the following strings is generated by (i.e., have a valid parse tree under) the grammar $S \rightarrow aSa \mid bSb \mid \epsilon$ (with start symbol S, and terminals a,b,ϵ)?

A. abSab
B. aabb
C. abba
D. abab
E. None of the above

CFG: Proving claims

- Since strings produced by a grammar are recursively defined, can often use induction to prove claims
  - e.g. $S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$
  - Claim: any string in this grammar's language is a palindrome
  - A string X, $|X|=n$ is a palindrome if for $i=1$ to n, $X[i] = X[n+1-i]$
  - Proof by induction on the height of the tree generating the string
  - Base case: h=1. Strings generated are a, b, ϵ, all palindromes ✔

CFG: Proving claims

For i=1 to n:=|X|, X[i] = X[n+1-i]

- S → aSa | bSb | a | b | ϵ
- Claim: any string in this grammar's language is a palindrome
- Base case: height=1. Strings generated are a, b, ϵ, all palindromes ✔
- Induction step: for all k ≥ 1,
  Hypothesis: suppose strings from trees of height ≤ k are palindromes
  To prove: Then, trees of height k+1 generate palindromes
  - Consider a tree with height k+1. Root has S→aSa or S→bSb.
  - String generated be X. Let |X|=n. X[1]=X[n] ✔
  - i.e., for i=1 and n, X[i]=X[(n+1)-i]. For i=2 to n-1?
  - Let Y be the string generated by the subtree rooted at the middle child of root, |Y|=n-2.
  - Y generated by a tree of height k. By IH, Y is a palindrome
    - For i=2 to n-1, X[i] = Y[i-1] = Y[(n-2)+1-(i-1)] = Y[n-i] = X[(n+1)-i] ✔

CFG: Proving claims

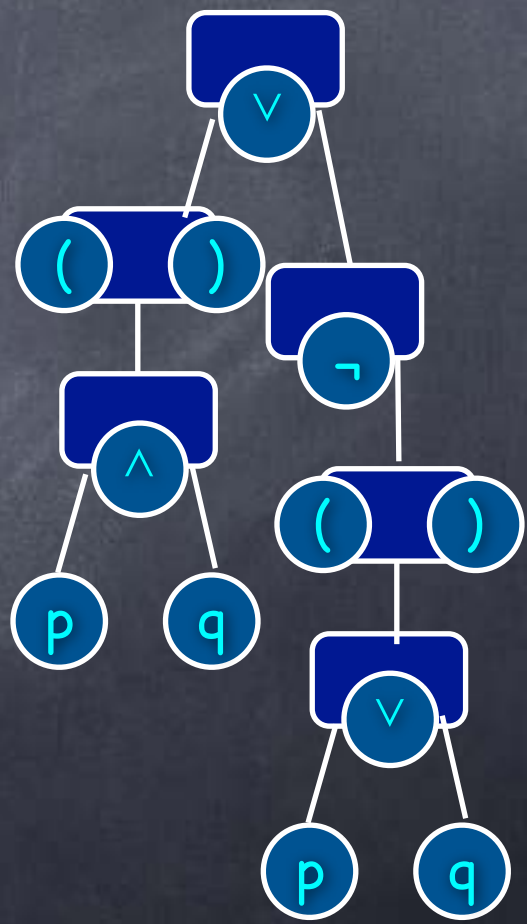
- Often prove a claim about all subtrees of trees generated by the grammar
  - With any non-terminal (or even terminal) at the root
  - Even if interested only in special cases (e.g. when root is a start symbol and leaves are all terminals)
- Recurring theme in proofs by induction: sometimes easier to prove stronger statements!

CFG: Proving claims

- e.g. $S \rightarrow AB \mid \epsilon$
  $A \rightarrow a \mid AS \mid SA$
  $B \rightarrow b \mid BS \mid SB$
  start symbol: S, terminals $\{a,b,\epsilon\}$
- Claim: Every string generated by the grammar has equal numbers of a's and b's
- **Stronger claim:** Every string generated by S has #a's = #b's, every string generated by A has #a's = #b's + 1 and every string generated by B has #b's = #a's+1
- By induction on the height of the grammar generating the string (starting with S, A or B at the root)

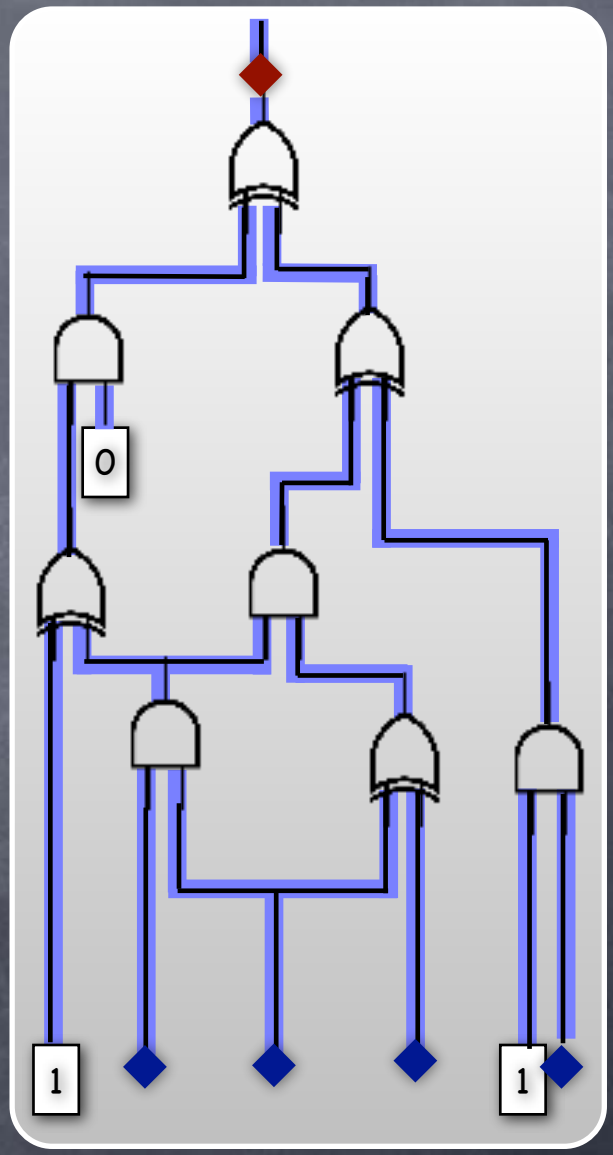
Formulas

- A recipe for creating a new proposition from given propositions
  - e.g. $f(p,q) \triangleq (p \wedge q) \vee \neg(p \vee q)$
- Exercise: A grammar for all formulas on a given set of variables
- The parse tree of a formula gives a way to evaluate the formula
  - A special case of a **circuit**



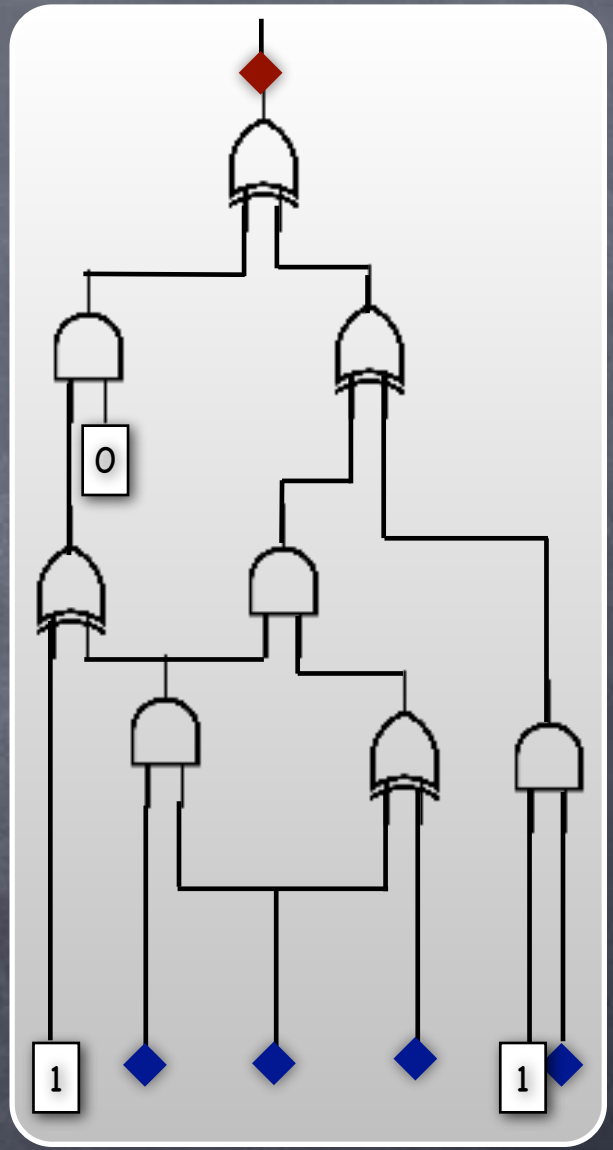
Circuits

- A circuit is a directed acyclic graph (DAG)
  - Edges: wires carrying values from a set. (e.g., boolean circuit: values in {0,1})
  - Nodes: Operator gates, constant gates, inputs, output(s)
    - e.g., for boolean circuits, operators can be AND, OR and NOT
      - May allow m-ary gates for AND etc.
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



Boolean Circuits

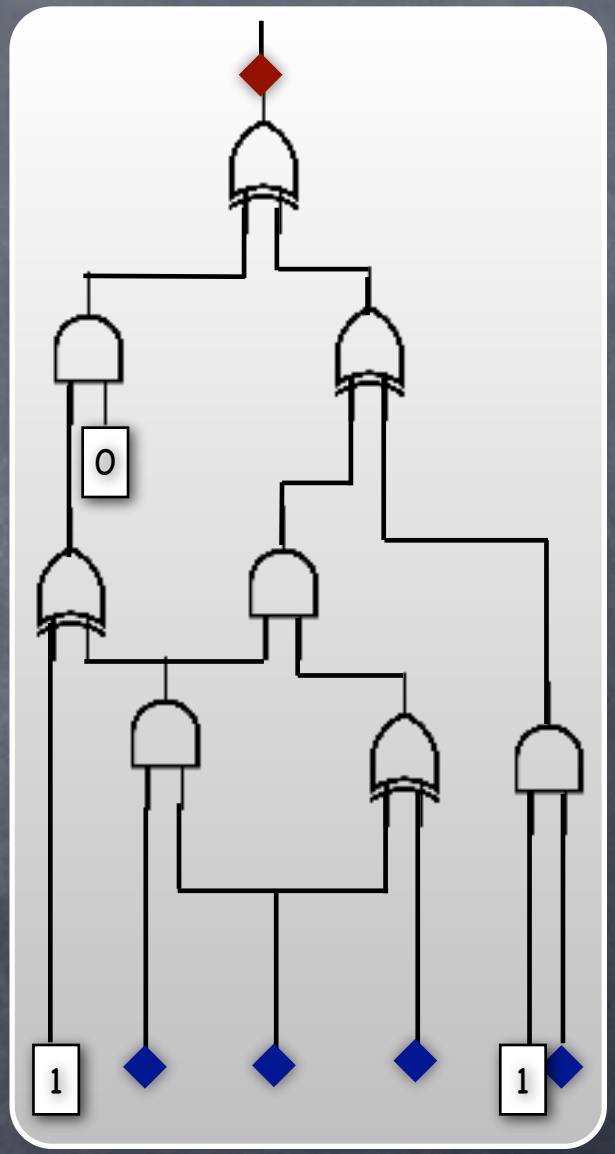
- A circuit can take an input of a fixed length only
- A circuit family: $(C_0, C_1, C_2, \ldots)$ where C_n takes inputs in $\{0,1\}^n$
  - A model for non-uniform computation
  - Quantities of interest (as a function of n): Circuit size (i.e., number of wires), and circuit depth



Can be improved to $O(2^n/n)$

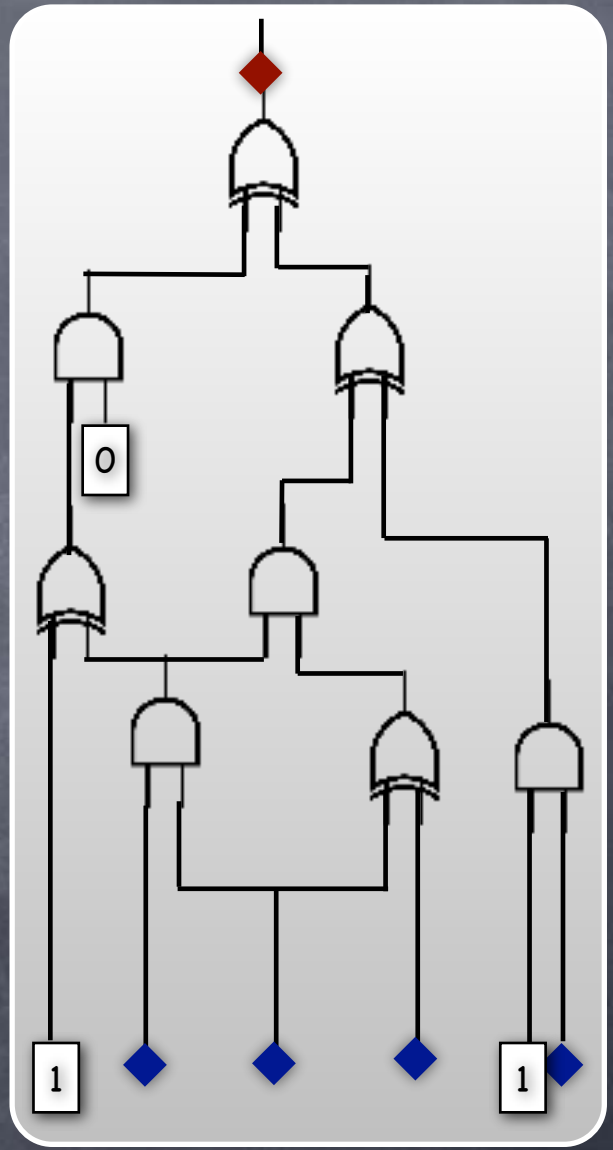
Boolean Circuits

- Every boolean function has a circuit family of size $O(2^n)$ and depth $O(1)$, with AND, OR and NOT gates
  - Let $S = \{ s \in \{0,1\}^n \mid f(s) = 1 \}$. $|S| \leq 2^n$.
  - Then $f(x) = \bigvee_{s \in S} (x=s)$
    $= \bigvee_{s \in S} \bigwedge_{i=1 \text{ to } n} (x_i=s_i)$
  - Circuit (in fact, formula):
    - $(x_i=1)$ and $(x_i=0)$ are x and $\neg x$. Use one n-ary AND gate for each $s \in S$, to check if $(x=s)$, and an $|S|$-ary OR gate as the output gate



Boolean Circuits

- Allowing m-ary gates not crucial
  - Exercise: implement an m-ary AND gate using a tree of binary AND gates
- With binary gates, circuit size typically defined as number of gates
- The exact choice of gates (AND, OR, NOT) not crucial
  - Exercise: implement each gate using
    - NAND gates alone
    - (AND, XOR) gates alone



A Lower Bound on Circuit Size

- Claim: Not all functions have circuits of size $\leq 2^n/(2n)$
- Proof: By counting the number of small circuits. W.l.o.g., use only binary NAND gates
- How many circuits with N gates (including input gates)?
  - Consider a topological sorting of the gates, with n input gates first and the output gate last. For i>n, i^{th} gate can choose its two inputs in $(i-1)^2$ ways. So, at most $[n \cdot (n+1) \cdot \ldots \cdot (N-1)]^2 \leq N^{2N}$ circuits
- How many functions? 2^{2^n}
- If all functions had size N circuits, then $N^{2N} \geq 2^{2^n}$
  - But if $N \leq 2^n/(2n)$, then $N^{2N} < (2^n)^{(2^n/n)} < 2^{2^n}$!