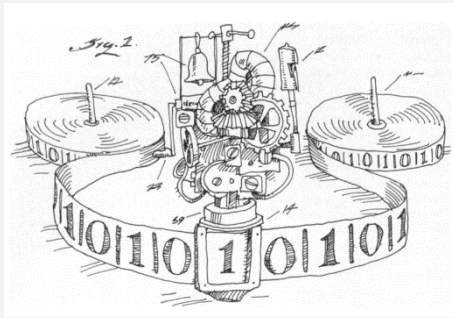
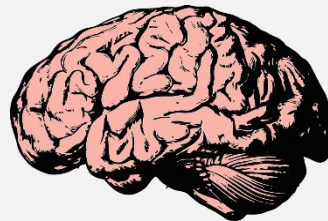
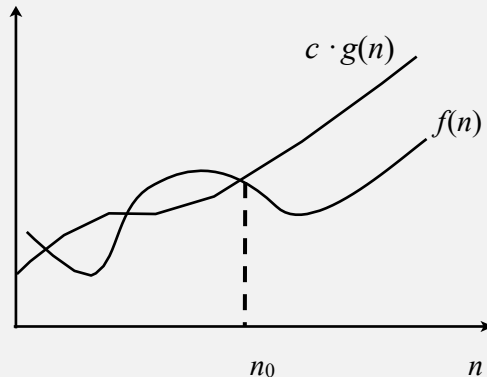


COMP 3602

Intro to Theory of Computation

Lecture 6



Logistics

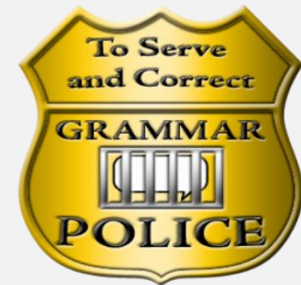
- Test 1 day/time: Mon Feb 2?
 - Written test on Turing machines, automata/PDA and today's material
 - Practice problems are up
- Theory reading group
 - Changing the time?
- Drills are up
 - Do the sample drills first!
 - You can do the drills multiple times, but the grade is an average of all attempts
 - So if you open it and let it time out, it will add a 0 to your average!

Questions/comments?

Formal grammars: Noam Chomsky

- Generative grammar: describes how to generate strings in a language
 - Similar idea to regular expressions.
 - Also called rewriting system
- Chomsky's grammars:
 - Alphabet Σ (called "*terminals*")
 - A set finite set V of *variables* (also called "non-terminals")
 - usually denoted by capital letters
 - A special *start variable* $S \in V$
 - A set of *production rules* R
 - Each rule is of the form $string_{left} \rightarrow string_{right}$, where $string_{left}, string_{right} \in (\Sigma \cup V)^*$
 - With an additional requirement that $string_{left}$ contains at least one variable symbol



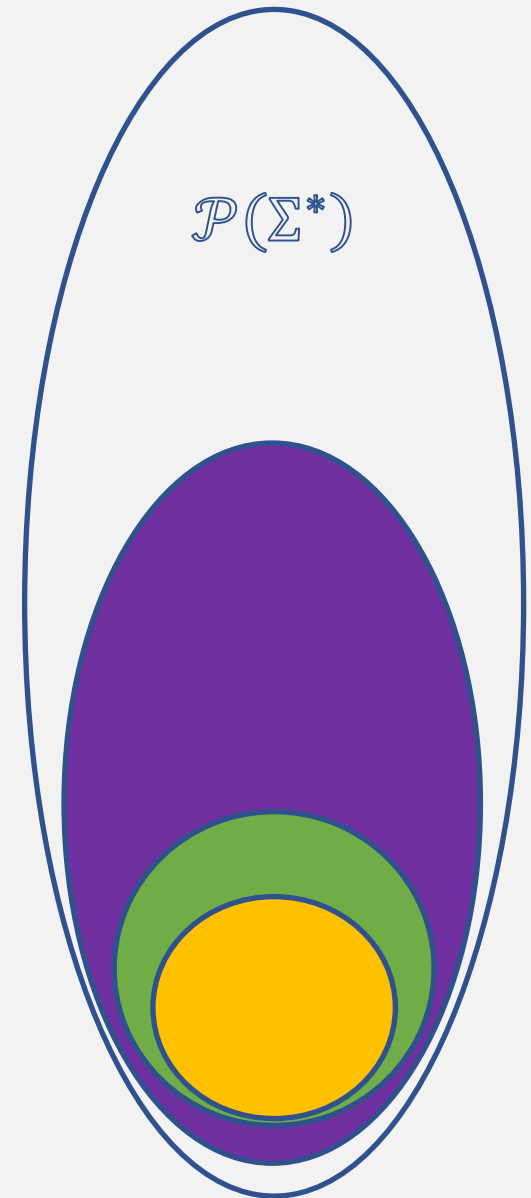


Derivation in a grammar

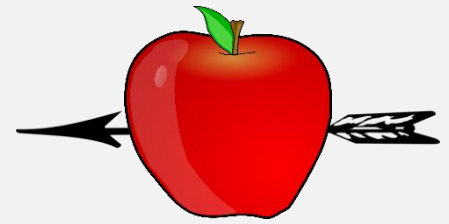
- Chomsky's grammars:
 - Alphabet Σ (called “*terminals*”)
 - A set finite set V of *variables* (also called “non-terminals”)
 - A special *start variable* $S \in V$
 - A set of *production rules* R
 - Each rule is of the form $string_{left} \rightarrow string_{right}$, where $string_{left}, string_{right} \in (\Sigma \cup V)^*$
 - $string_{left}$ contains at least one variable symbol
- A **derivation** is a sequence of strings each of which is obtained from the previous by applying some rule to its substring.
 - Example: can apply the rule $S \rightarrow 0S$ to a string $01S0S1$ once to get $010S0S1$
 - Can also apply the rule to the second S to get $01S00S1$
- A language generated by a grammar is all strings over Σ that can be derived starting from S .

Chomsky's hierarchy

- **Unrestricted (including context-sensitive)** grammars
 - Generate exactly the languages recognized by Turing machines.
 - Left string is a combination of variables and terminals with at least one variable
- **Context-free** grammars:
 - Left string in rules is always a single variable
 - Rules of the form $A \rightarrow string$ for $A \in V$, $string \in (\Sigma \cup V)^*$
 - Strictly stronger than regular, strictly weaker than context-sensitive
 - Generate the class of languages recognized by pushdown automata!
- **Regular** grammar
 - Rules of the form $A \rightarrow sB$ or $A \rightarrow \lambda$ where $s \in \Sigma$ and $A, B \in V$
 - Think of variables corresponding to states, s an input symbol being read
 - Generate regular languages



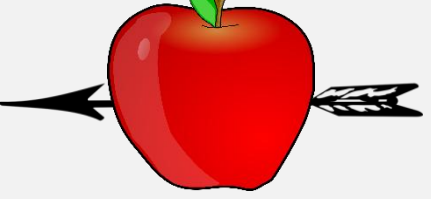
Context-free grammars



A special kind of grammars is **context-free grammars**, where symbols have the same meaning wherever they are.

A context-free grammar consists of

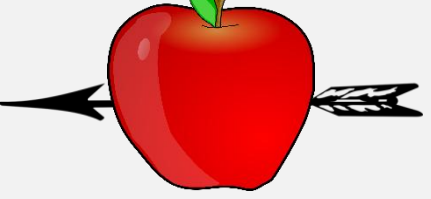
- A set V of **variables** (using capital letters), including a **start variable** S .
 - Note: these are variables, not sets.
- A set Σ of **terminals** (disjoint from V ; alphabet of the language.)
- A set R of **rules**, where each rule consists of a variable from V , followed by $\rightarrow$, followed by a string of variables and terminals.
 - This is not the same “ $\rightarrow$ ” as implication, a different use of the same symbol.
 - If several rules start with the same non-terminal such as $A \rightarrow w_1$, $A \rightarrow w_2$, etc, we use a shortcut “ $|$ ” (means “or”) to write them as one rule: $A \rightarrow w_1 | w_2 | \dots | w_k$



Context-free grammars

A context-free grammar: a set V of **variables**, including a **start variable** S , a set Σ (Sigma) of **terminals** (alphabet) and a set R of **rules** of the form $A \rightarrow w$, where $A \in V, w \in (V \cup \Sigma)^*$

- If $A \rightarrow w$ is a rule, we say variable A **yields** string w .
 - Can use A within the rule, as many times as we want: recursion!
 - Different occurrences of the same variable can produce different strings.
- A **derivation** is a sequence of strings each of which is obtained from the previous by applying some rule to its substring.
 - If $w = \ell Ar$ and there is a rule $A \rightarrow u$, then ℓur is **directly derived** from w , written $w \Rightarrow \ell ur$.
 - w_n is **derived from** w_0 if there is a sequence $w_0, w_1, \dots, w_n$ where $\forall i \ w_i \Rightarrow w_{i+1}$
- A **language generated by a grammar** consists of all strings of terminals that can be derived from the start variable S .



Context-free grammars

A context-free grammar: a set V of **variables**, including a **start variable** S , a set Σ (Sigma) of **terminals** (alphabet) and a set R of **rules** of the form $A \rightarrow w$, where $A \in V, w \in (V \cup \Sigma)^*$

- **A language generated by a grammar** consists of all strings of terminals that can be derived from the start variable S .
- A language is context-free (CFL) iff it is generated by a some CFG
 - All regular languages are context-free.
 - Some context-free languages are not regular

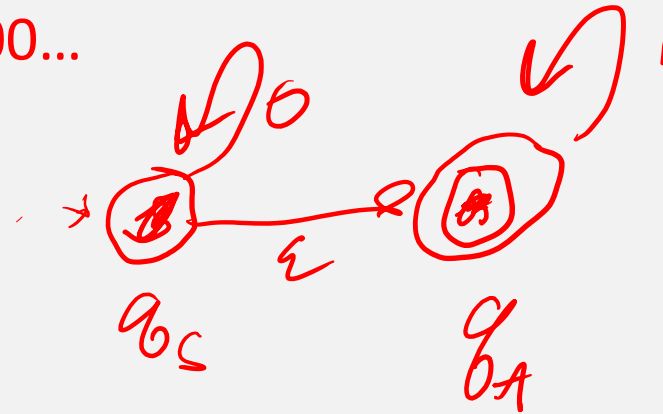
Example: language L of all strings over $\{0,1\}$ with all 0s before all 1s.

Strings in L : $\lambda, 0, 1, 11, 001, 00111, 011111, 00000000, \dots$

Strings not in L : $10, 1110, 010101, 00100000, \dots$

– Regular expression:

0^*1^*



$S \rightarrow \lambda$

$S \rightarrow 0S$

$S \rightarrow S1$

- Context-free grammar for L
 - 3 rules: $S \rightarrow 0S, S \rightarrow S1, S \rightarrow \lambda$
 - Shorter description of the grammar: $S \rightarrow 0S \mid S1 \mid \lambda$
 - Variables: $V = \{S\}$. Terminals: $\Sigma = \{0,1\}$
 - Derivation of a string 001: $S \Rightarrow 0S \Rightarrow 0S1 \Rightarrow 00S1 \Rightarrow 00\lambda1 = 001$
 - Alternative derivation of 001: $S \Rightarrow S1 \Rightarrow 0S1 \Rightarrow 00S1 \Rightarrow 00\lambda1 = 001$
- Regular grammar?
 - Rules: $S \rightarrow 0S \mid A, A \rightarrow 1A \mid \lambda$



Parse trees:

- Parse trees: visualizing derivations
 - Similar to syntax trees,
 - except all internal nodes are variables,
 - and all nodes on the bottom are terminals.

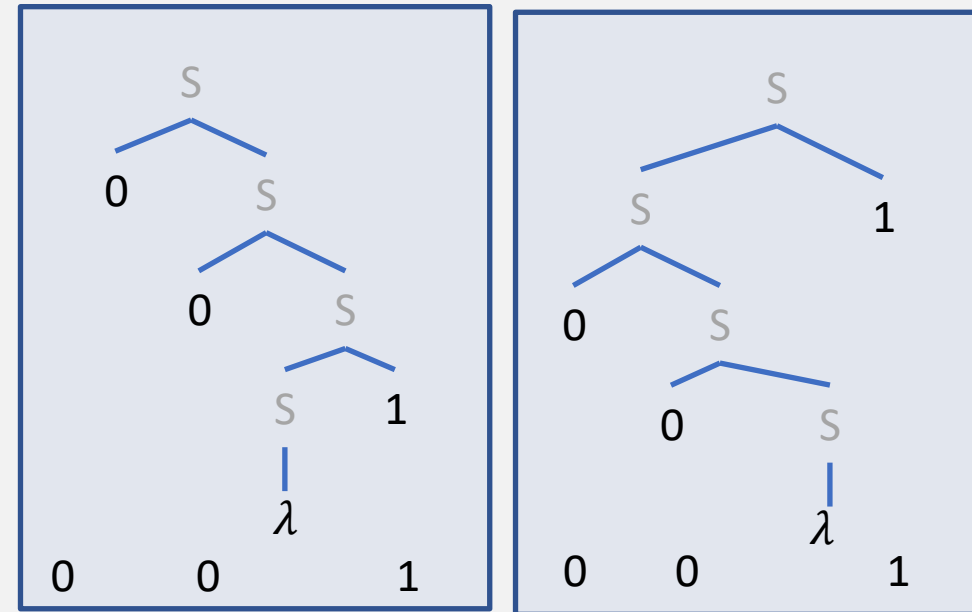
Grammar: $S \rightarrow 0S \mid S1 \mid \lambda$

- Derivation of a string 001:

$S \Rightarrow 0S \Rightarrow 00S \Rightarrow 00S1 \Rightarrow 00\lambda 1 = 001$

- Alternative derivation of 001:

$S \Rightarrow S1 \Rightarrow 0S1 \Rightarrow 00S1 \Rightarrow 00\lambda 1 = 001$



Examples of context-free grammars

- L contains two strings 1 and 00

$$S \rightarrow 1 \mid 00$$

- Variables: S.
- Terminals: 1 and 00 (or 1 and 0).

- L is strings over $\{a, b, c, d\}$ with even number of “a”s

$$S \rightarrow SaSaS \mid SbS \mid ScS \mid SdS \mid \lambda$$

- Variables: S. Terminals: a,b,c,d.

- $L = \{0^n 1^n\}$

$$S \rightarrow 0S1 \mid \lambda$$

- Variables: S. Terminals: $\{0,1\}$.

Not regular!

- Natural numbers, start symbol N

$$N \rightarrow 0 \mid M$$

$$M \rightarrow 1D \mid 2D \mid 3D \mid 4D \mid 5D \mid 6D \mid 7D \mid 8D \mid 9D$$

$$D \rightarrow M \mid 0D \mid \lambda$$

- Rational numbers: same as natural numbers plus a new start symbol R, new variable P and new rules

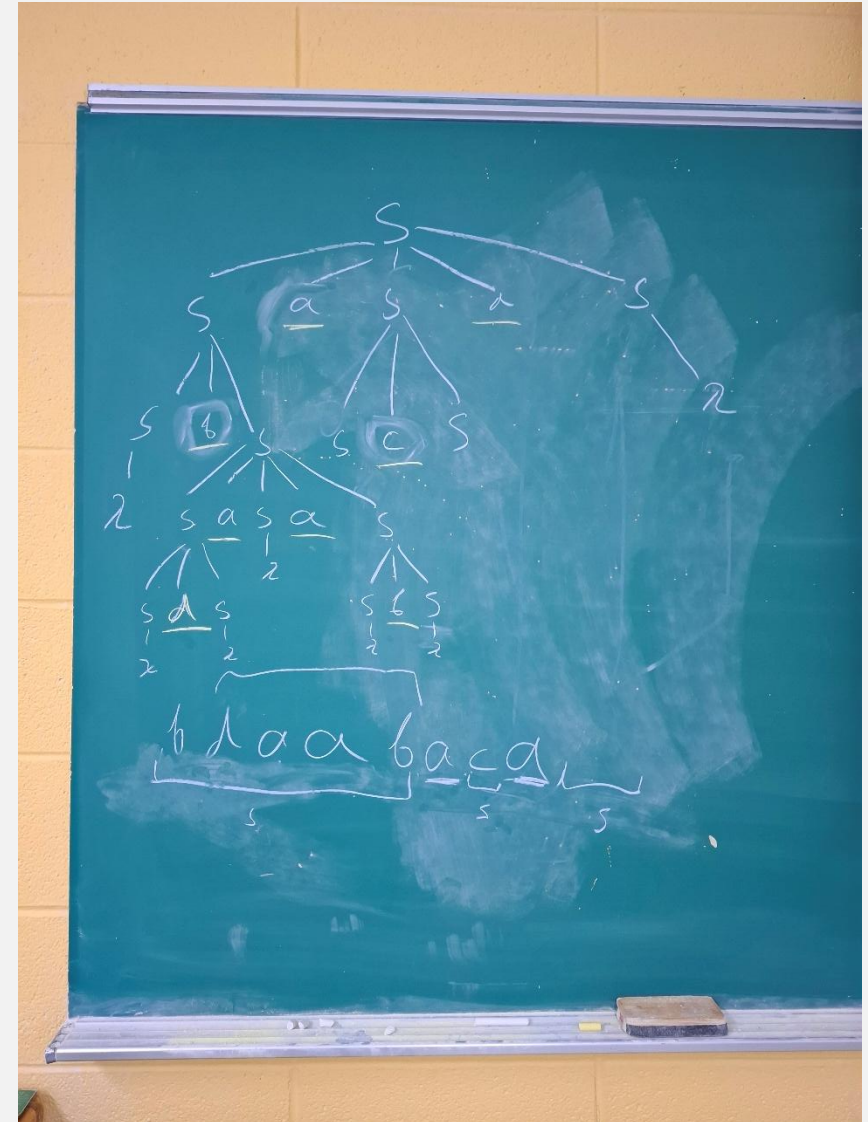
$$R \rightarrow P \mid -P$$

$$P \rightarrow N \mid N/M \mid N.N$$

- L is strings over $\{a, b, c, d\}$ with even number of "a"s

$$S \rightarrow SaSaS \mid SbS \mid ScS \mid SdS \mid \lambda$$

– Variables: S. Terminals: a,b,c,d.

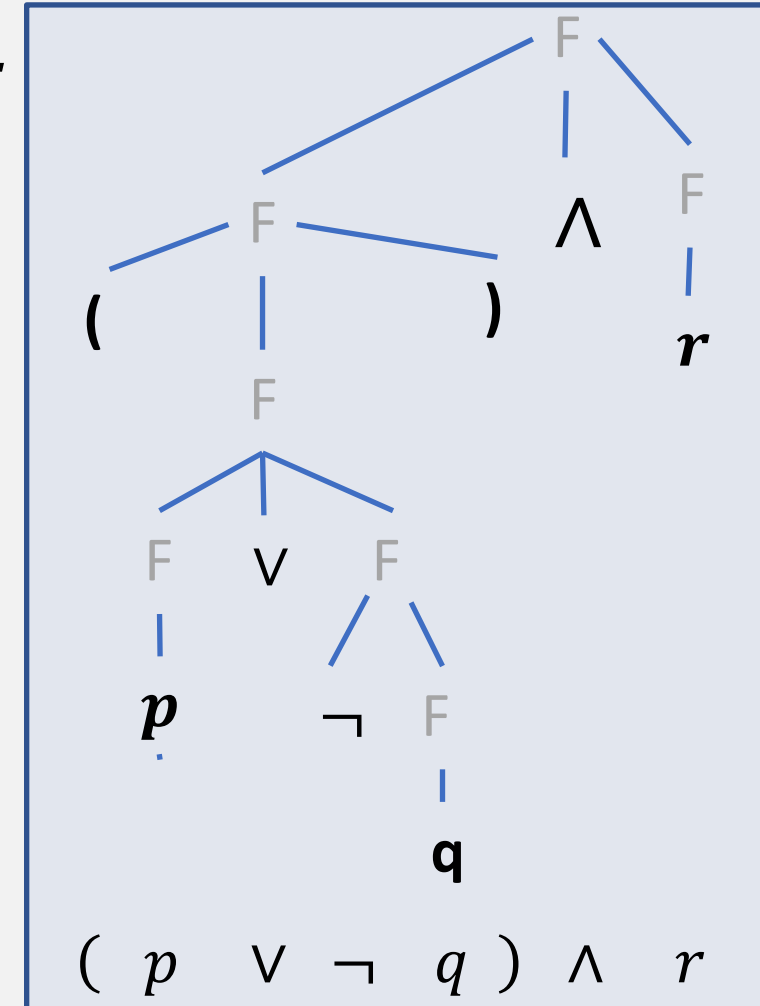


Propositional formulas

- Recursive definition:
 - Basis: a propositional variable p, q, r or a constant $TRUE, FALSE$
 - Recursion: if F and G are propositional formulas, then so are (F) , $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$, $F \leftrightarrow G$.
- Grammar: $F \rightarrow F \vee F | F \wedge F | \neg F | (F) | F \rightarrow F | F \leftrightarrow F$
 $F \rightarrow p | q | r | TRUE | FALSE$

Here, the only variable is F (it is a start variable), and the set of terminals is $\{\vee, \wedge, \neg, \rightarrow, \leftrightarrow, (,), p, q, r, TRUE, FALSE\}$

Deriving $(p \vee \neg q) \wedge r$: $F \Rightarrow F \wedge F \Rightarrow (F) \wedge F \Rightarrow (F) \wedge r \Rightarrow (F \vee F) \wedge r \Rightarrow (p \vee F) \wedge r \Rightarrow (p \vee \neg F) \wedge r \Rightarrow (p \vee \neg q) \wedge r$



Context-free grammars for arithmetic expressions

$EXPR \rightarrow EXPR + EXPR \mid EXPR - EXPR \mid EXPR * EXPR$

$EXPR \rightarrow EXPR / EXPR \mid (EXPR) \mid 0 \mid NUMBER \mid -NUMBER$

$NUMBER \rightarrow 1DIGITS \mid \dots \mid 9DIGITS$

$DIGITS \rightarrow \lambda \mid NUMBER \mid 0DIGITS$

Grammar for natural numbers

– Variables: $EXPR$, $NUMBER$, $DIGITS$ ($EXPR$ is starting).

– Terminals: $+$, $-$, $*$, $/$, $0, \dots, 9, (,)$.

$5 * 3 - 4$

$5 + 3 / 4$

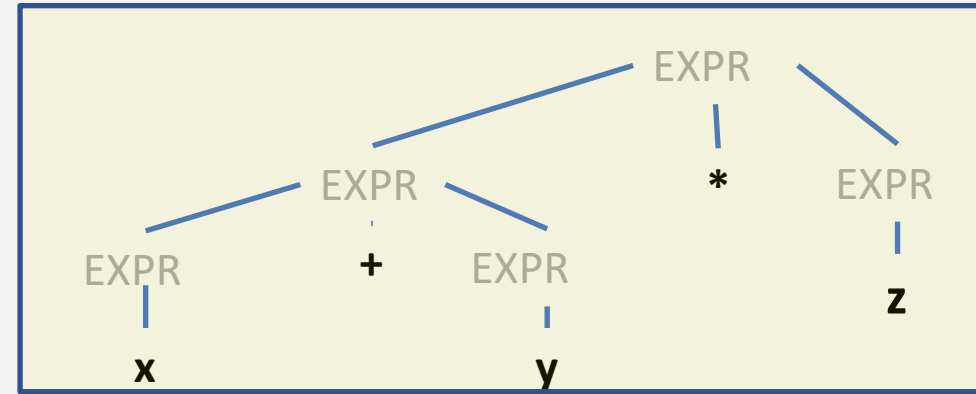
- Problem: this definition of arithmetic expressions (and the previous definition of propositional formulas) do not have any information about order of operations.

Encoding order of precedence

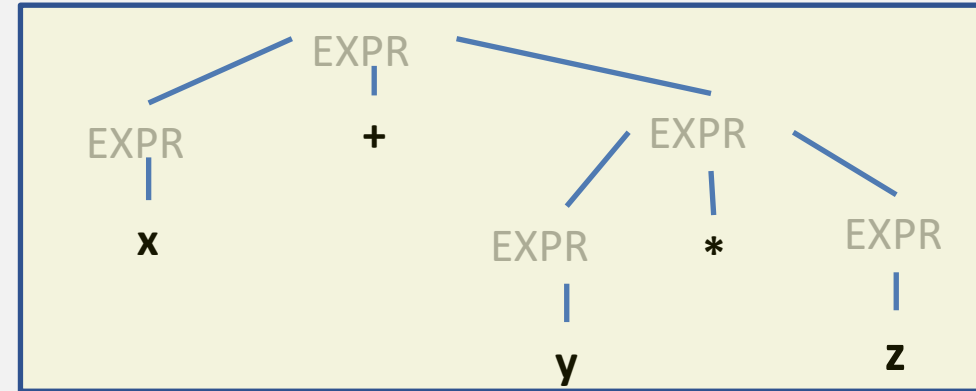
- Easier to specify in which order to process parts of the formula.
 - Better grammar for arithmetic expressions (for simplicity, with only x,y,z instead of numbers):
$$\begin{aligned} \textit{EXPR} &\rightarrow \textit{EXPR} + \textit{TERM} \mid \textit{EXPR} - \textit{TERM} \mid \textit{TERM} \\ \textit{TERM} &\rightarrow \textit{TERM} * \textit{FACTOR} \mid \textit{TERM} / \textit{FACTOR} \mid \textit{FACTOR} \\ \textit{FACTOR} &\rightarrow (\textit{EXPR}) \mid x \mid y \mid z \end{aligned}$$
 - Here, variables are EXPR, TERM and FACTOR (with EXPR a starting variable).
 - Now can encode precedence.

Ambiguous grammars

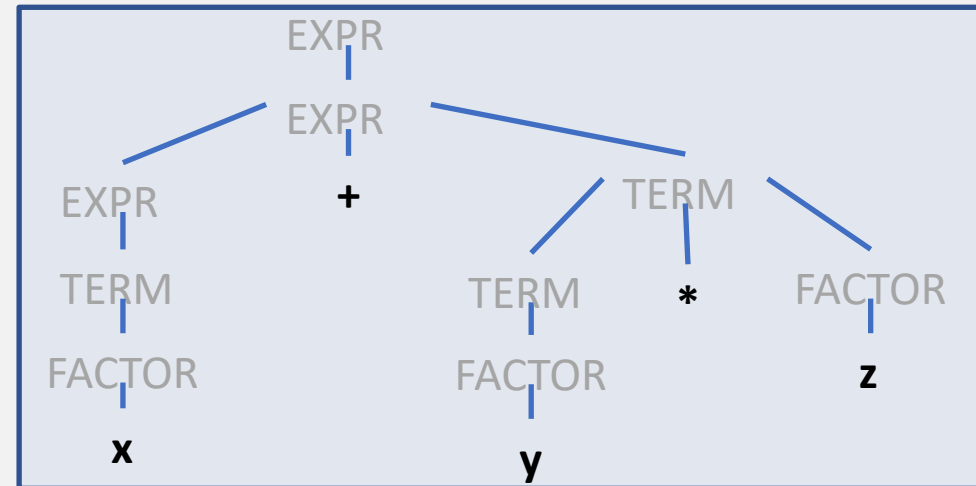
- A context-free grammar is **ambiguous** if there is some string with more than one possible parse tree.
 - Grammar $S \rightarrow 0S|S1|\lambda$ was ambiguous since 001 had two parse trees.



- First grammar for arithmetic expressions (and grammar for propositional formulas) were ambiguous: $x + y * z$ has two parse trees.

$$EXPR \rightarrow EXPR + EXPR \mid EXPR - EXPR$$
$$EXPR \rightarrow EXPR * EXPR \mid EXPR / EXPR$$
$$EXPR \rightarrow (EXPR)|x|y|z$$


- Second grammar for arithmetic expressions is not ambiguous

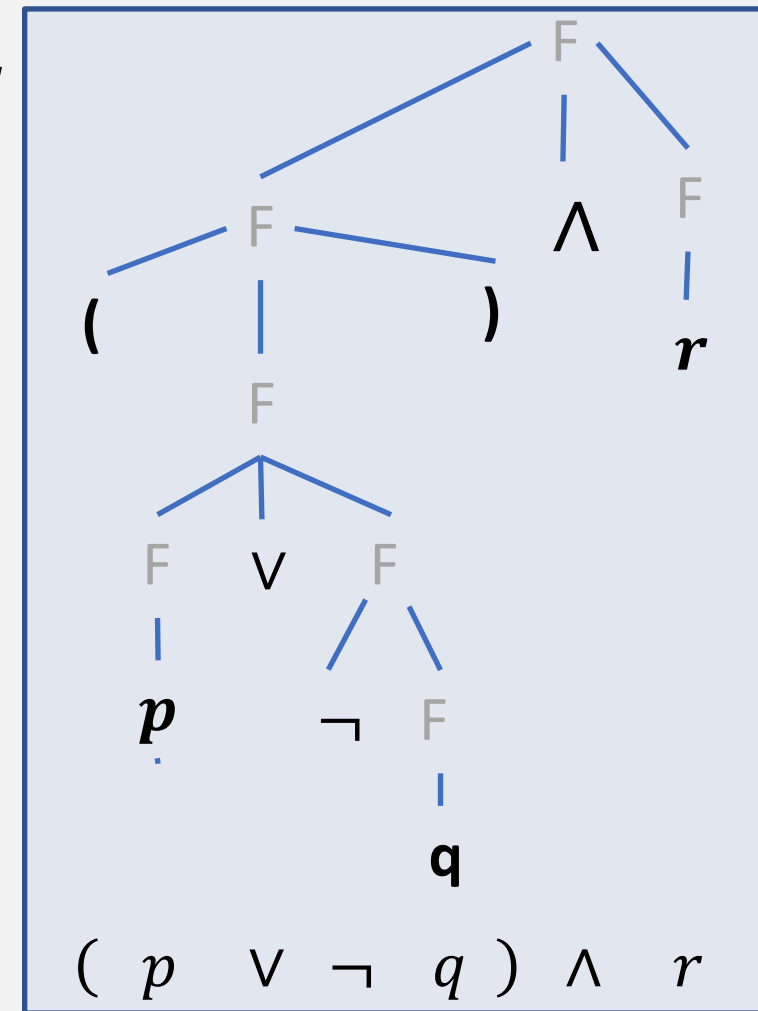
$$EXPR \rightarrow EXPR + TERM \mid EXPR - TERM \mid TERM$$
$$TERM \rightarrow TERM * FACTOR \mid TERM / FACTOR \mid FACTOR$$
$$FACTOR \rightarrow (EXPR) \mid x \mid y \mid z$$


Parse trees

- Propositional formulas:
 - Basis: a propositional variable p, q, r or a constant $TRUE, FALSE$
 - Recursion: if F and G are propositional formulas, then so are (F) , $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$, $F \leftrightarrow G$.
- Grammar: $F \rightarrow F \vee F \mid F \wedge F \mid \neg F \mid (F) \mid F \rightarrow G \mid F \leftrightarrow G$
 $F \rightarrow p \mid q \mid r \mid TRUE \mid FALSE$

Here, the only variable is F (it is a start variable), and the set of terminals is $\{V, \wedge, \neg, \rightarrow, \leftrightarrow, (,), p, q, r, TRUE, FALSE\}$

Deriving $(p \vee \neg q) \wedge r$: $F \Rightarrow F \wedge F \Rightarrow (F) \wedge F \Rightarrow (F) \wedge r \Rightarrow$
 $\Rightarrow (F \vee F) \wedge r \Rightarrow (p \vee F) \wedge r \Rightarrow (p \vee \neg F) \wedge r \Rightarrow (p \vee \neg q) \wedge r$



Pumping lemma for context-free languages

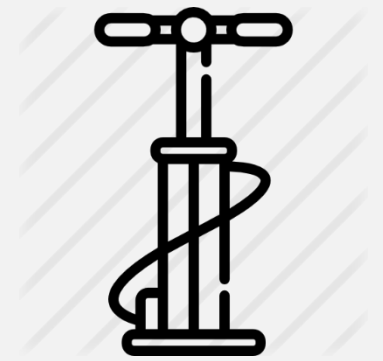
- *Pumping lemma:* If L is a **context-free** language, then there exists a number p (pumping length) such that every string s with $|s| \geq p$ can be written as a concatenation of strings u, v, x, y, z ($s = uvxyz$) where
 1. For each $i \geq 0$, string $uv^i xy^i z \in L$
 2. $|vy| > 0$
 3. $|vxy| \leq p$

Writing the statement with quantifiers:

$\exists p \in \mathbb{N} \ \forall s \in L (|s| \geq p \rightarrow$

$\exists u, v, x, y, z \ s = uvxyz \wedge \forall i \ uv^i xy^i z \in L \wedge |vy| > 0 \wedge |vxy| \leq p)$

Proof idea: a non-terminal must repeat on a path when the parse tree is large enough .





- To prove that L is not context-free
- We pick a string $s \in L$ that depends on a parameter p
- Show that for any split of s in u, v, x, y, z such that $|vy| > 0$ and $|vxy| \leq p$, there is an $i \geq 0$ with $uv^i xy^i z \notin L$
- Example: $\{a^n b^n c^n \mid w \in \{a, b, c\}^*\}$
 - Let $s = a^p b^p c^p$
 - Similar to $0^p 1^p$ for proving $\{0^n 1^n\}$ is non-regular
 - Let $s = uvxyz$, with $|vy| > 0$
 - Case 1: each of v, y contains at most one type of symbol (all a's, all b's, all c's)
 - Then pumping does not change the third type
 - Case 2: at least one of v, y contains a mix
 - Then $uv^2 xy^2 z$ has symbols in wrong order.



- So $\{a^n b^n c^n \mid w \in \{a, b, c\}^*\}$ is not context-free
 - But $\{a^i b^j c^k \mid i = j\}$ and $\{a^i b^j c^k \mid i = k\}$ are context-free!
- *Corollary:* Context-free languages are not closed under intersection and complementation.
 - Proof: for intersection, the counterexample above.
 - For complementation:
 - They are closed under union (do ε -arrows at the new start, as for regular)
 - If they were closed under complementation, could have done DeMorgan's $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ to get intersection. But we know we cannot get intersection by above.



- To prove that L is not context-free
 - We pick a string $s \in L$ that depends on a parameter p
 - Show that for any split of s in u, v, x, y, z such that $|vy| > 0$ and $|vxy| \leq p$, there is an $i \geq 0$ with $uv^i xy^i z \notin L$
-
- Example: $\{ww \mid w \in \{0,1\}^*\}$
 - Let $s = 0^p 1^p 0^p 1^p$
 - Before we used $0^p 10^p 1$. But it does not work! With $v = 0, x = 1, y = 0$ can pump.
 - Let $s = uvxyz$, with $|vy| > 0$
 - vxy should have the middle of the string
 - If vxy are all in the first half, then pumping it would move 1s in front of 2nd w.
 - Or make the first w only 0s, while second half has 1s.
 - Same for in the second half.
 - Suppose we pump down s ; since $|vxy| \leq p$, get $0^p 1^i 0^j 1^p$ with $i \neq p$ or $j \neq p$
 - Thus this string is not of the form ww .

Pushdown automata: NFA+stack



- Pushdown automaton (PDA):
 - Input alphabet Σ
 - **Stack alphabet Γ**
 - Set of states Q
 - Start state $q_0 \in Q$
 - Set of final (accepting) states $F \subseteq Q$
 - Transition function $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$
- Computation of a PDA M on a string $w = w_1 \dots w_n$:
 - On each step M reads a symbol from the input, pops a symbol off the stack and then changes state and pushes a symbol on the stack
 - In all three of read, pop, push, the “symbol” may be empty, that is, ε .
 - Popping an empty stack, in particular, always returns ε
 - Often want to put an end marker $\$$ into the stack in the first transition to detect empty.
 - M accepts w if there exist sequences of states $r_0 \dots r_m$ **and stack content strings** $s_0 \dots s_m$ with $m \geq n$, $r_0 = q_0$, $s_0 = \lambda$, $r_m \in F$ and all intermediate states and stacks computed correctly.

Context-free languages = recognized by PDAs

- From a grammar to a PDA:
 - Match terminals right away
 - Keep the remaining string (starting with first non-terminal) on the stack
 - Guess what rule to use with non-determinism

From PDA to grammar

- Idea: make a variable for every pair of states: $V = \{A_{pq} \mid p, q, \in Q\}$
 - It will generate strings processed by the PDA between these states

From a PDA to a grammar

- Let M be a PDA.  Without loss of generality
 - Assume WLOG that each transition either pushes or pops, but not both.
 - And that $|F| = \{q_a\}$, and at accept the stack is empty.
- Make a variable for every pair of states: $V = \{A_{pq} \mid p, q \in Q\}$
- Start variable A_{q_0, q_a}
- Rules:
 - 1. Split into separate strings: for every triple of states add a rule $A_{pq} \rightarrow A_{pr}A_{rq}$
 - When the stack is same at all three of p, q, r
 - 2. Push/pop pair: If $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$, add $A_{pq} \rightarrow aA_{rs}b$
 - 3. Add $A_{pp} \rightarrow \lambda$ for all $p \in Q$