Computational Models - Lecture 3

- Non Regular Languages: Two Approaches
- (1) The Pumping Lemma
- (2) Myhill-Nerode Theorem (not in Sipser's book)
- Closure properties
- Algorithmic questions for NFAs

- Sipser's book, 1.4, 2.1, 2.2
- Hopcroft and Ullman, 3.4

Proved Last Time

Thm.: A language, L, is described by a regular expression, R, if and only if L is regular.

$\Longrightarrow$ construct an NFA accepting R.

$\Longleftarrow$ Given a regular language, L, construct an equivalent regular expression

Negative Results

We have made a lot of progress understanding what finite automata can do. But what can't they do?
Is there a DFA that accepts

- $B = \{0^n 1^n | n \geq 0\}$
- $C = \{w | w$ has an equal number of 0's and 1's$\}$
- $D = \{w | w$ has an equal number of occurrences of 01 and 10 substrings$\}$

Consider B:

- DFA must "remember" how many 0's it has seen
- impossible with finite state.

The others are exactly the same.

Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n | n \geq 0\}$
- $C = \{w | w$ has an equal number of 0's and 1's$\}$
- $D = \{w | w$ has an equal number of occurrences of 01 and 10 substrings$\}$

Consider B:

- DFA must "remember" how many 0's it has seen
- impossible with finite state.

The others are exactly the same...

Question: Is this a proof?
Answer: No, D is regular!???

Pumping Lemma

We will show that all regular languages have a special property.

- Suppose L is regular.
- If a string in L is longer than a certain critical length ℓ (the pumping length),
- then it can be "pumped" to a longer string by repeating an internal substring any number of times.
- The longer string must be in L too.
- This is a powerful technique for showing that a language is not regular.

Pumping Lemma

Theorem: If L is a regular language, then there is an $\ell > 0$ (the pumping length), where if s is any string in L of length $|s| \geq \ell$, then s may be divided into three pieces $s = xyz$ such that

- for every $i \geq 0$, $xy^iz \in L$,
- $|y| > 0$, and
- $|xy| \leq \ell$.

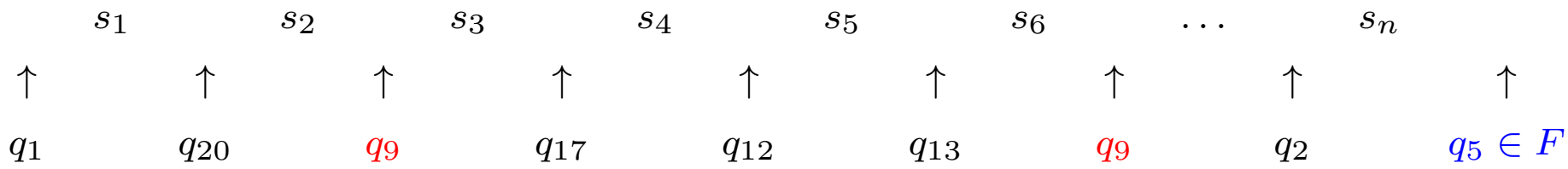
Remarks: Without the second condition, the theorem would be trivial.
The third condition is technical and sometimes useful.

Pumping Lemma – Proof

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that accepts L.

Let ℓ be $|Q|$, the number of states of M.

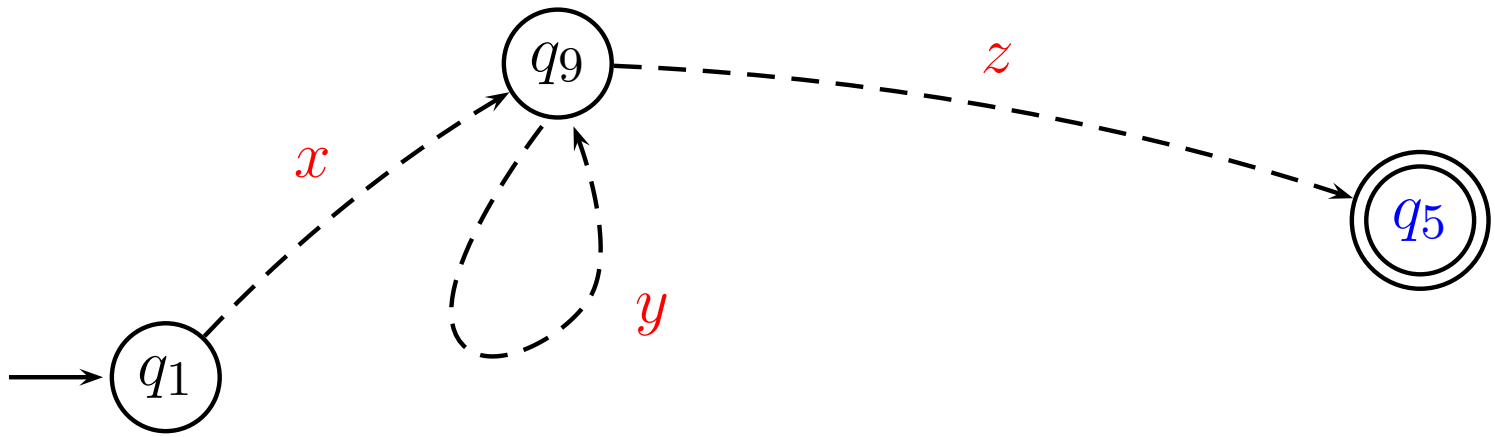
If $s \in L$ has length at least ℓ, consider the sequence of states M goes through as it reads s:



Since the sequence of states is of length $|s| + 1 > \ell$, and there are only ℓ different states in Q, at least one state is repeated (by the pigeonhole principle).

Pumping Lemma – Proof (cont.)

Write down $s = xyz$



- By inspection, M accepts xy^kz for every $k \geq 0$.
- $|y| > 0$ because the state (q_9 in figure) is repeated.
- To ensure that $|xy| \leq \ell$, pick first state repetition, which must occure no later than $\ell + 1$ states in sequence.

An Application

Theorem: The language $B = \{0^n 1^n | n > 0\}$ is not regular.

Proof: By contradiction. Suppose B is regular, accepted by DFA M. Let 2ℓ be the pumping length.

- Consider the string $s = 0^\ell 1^\ell$.
- By pumping lemma $s = xyz$, where $xy^k z \in B$ for every k.
- If y is all 0, then $xy^k z$ has too many 0's.
- If y is all 1, then $xy^k z$ has too many 1's.
- If y is mixed, then $xy^k z$ is not of right form. ♣

Using the Pumping Lemma

How to prove that a language is not regular

- Select a language L
- An adversary sets the parameter ℓ.
- Select a word $s \in L$.
  - The word s would (usually) depend on ℓ.
- The adversary selects a partition $s = xyz$, such that $|y| \geq 1$ and $|xy| \leq \ell$.
- Show an index k, such that $xy^k z \notin L$.
  - Need to prove for any parameter ℓ and any partition xyz.

Another Application

Theorem: The language
$C = \{w | w$ has an equal number of 0's and 1's$\}$
is not regular.

Proof: By contradiction. Suppose C is regular, accepted by DFA M. Let ℓ be the pumping length.

- Consider the string $s = 0^\ell 1^\ell$.
- By pumping lemma $s = xyz$, where $xy^k z \in C$ for every k.
- Since $|xy| \leq \ell$ then y is all 0, and $xy^k z$ has too many 0's.

♣

Yet Another Application

Theorem: The language $Primes \subset \{1\}^*$, which contains all strings whose length is a prime number, is not regular.

Proof: By contradiction. Suppose $Primes$ is regular, accepted by DFA M. Let ℓ be the pumping length.

- Let $s = 1^p \in Primes$, where $p \geq \ell$ is a prime.
- By pumping lemma $s = xyz$, where $xy^kz \in Primes$ for every k.
- For $k = p + 1$ we have $xy^kz = 1^{p+mp}$, where $|y| = m$.
- since $p(m + 1)$ is not prime, we have a contradiction. ♣

Another example

consider the language
$L = \{a^i b^n c^n | n \geq 0, i \geq 1\} \cup \{b^n c^m | n, m \geq 0\}$,

For any word $s \in L$ we can apply the pumping lemma:

- If $s = a^i b^n c^n$, then set $x = \varepsilon$ and $y = a$.
- If $s = b^n c^m$, then set $x = \varepsilon$ and $y = b$.
- Is L regular?!
- How can we prove it?!

Characterization of Regular Languages

The Equivalence Relation $\sim_L$

Let $L \subseteq \Sigma^*$ be a language.

Define an equivalence relation $\sim_L$ on pairs of strings:

Let $x, y \in \Sigma^*$. We say that $x \sim_L y$ if for every string $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$.

It is easy to see that $\sim_L$ is indeed an equivalence relation (reflexive, symmetric, transitive) on Σ^*.

In addition, if $x \sim_L y$ then for every string $z \in \Sigma^*$, $xz \sim_L yz$ as well (this is called right invariance).

The Equivalence Relation $\sim_L$

Like every equivalence relation, $\sim_L$ partitions Σ^* to (disjoint) equivalence classes. For every string x, let $[x] \subseteq \Sigma^*$ denote its equivalence class w.r.t. $\sim_L$ (if $x \sim_L y$ then $[x] = [y]$ – equality of sets).

Question is, how many equivalence classes does $\sim_L$ induce?

In particular, is the number of equivalence classes of $\sim_L$ finite or infinite?

Well, it could be either finite or infinite. This depends on the language L.

Classes of $\sim_L$: Three Examples

- Let $L_1 \subset \{0,1\}^*$ contain all strings where the number of 1s is divisible by 4. Then $\sim_{L_1}$ has finitely many equivalence classes.
- Let $L_2 \subset \{0,1\}^*$ contain all strings of the form $0^n 1^n$. Then $\sim_{L_2}$ has infinitely many equivalence classes.
- Let $L_3 = \{a^i b^n c^n | n \geq 0, i \geq 1\} \cup \{b^n c^m | n, m \geq 0\}$. Then $\sim_{L_3}$ has infinitely many equivalence classes.

Myhill-Nerode Theorem

Theorem: Let $L \subseteq \Sigma^*$ be a language. Then

L is regular $\iff$ $\sim_L$ has finitely many equivalence classes.

- Three specific consequences:
- $L_1 \subset \{0,1\}^*$ contains all strings where the number of 1s is divisible by 4. Then L_1 is regular.
- $L_2 \subset \{0,1\}^*$ contains all strings of the form $0^n 1^n$. Then L_2 is not regular.
- Let $L_3 = \{a^i b^n c^n | n \geq 0, i \geq 1\} \cup \{b^n c^m | n, m \geq 0\}$. Then L_3 is not regular.

Myhill-Nerode Theorem: Proof

$\Longrightarrow$

Suppose L is regular. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting it. For every $x \in \Sigma^*$, let $\alpha(q_0, x) \in Q$ be the state where the computation of M on input x ends.

The relation $\sim_M$ on pairs of strings is defined as follows: $x \sim_M y$ if $\alpha(q_0, x) = \alpha(q_0, y)$.
Clearly, $\sim_M$ is an equivalence relation.

Furthermore, if $x \sim_M y$, then for every $z \in \Sigma^*$, also $xz \sim_M yz$. Therefore, $xz \in L$ if and only if $yz \in L$.

This means that $x \sim_M y \Longrightarrow x \sim_L y$.

Myhill-Nerode Theorem: Proof (cont.)

$\Longrightarrow$

- The equivalence relation $\sim_M$ has finitely many equivalence classes (at most the number of states in M).
- We saw that $x \sim_M y \Longrightarrow x \sim_L y$, so the number of equivalence classes of $\sim_L$ is less or equal than the number of equivalence classes of $\sim_M$.
- Therefore, $\sim_L$ has finitely many equivalence classes. ♠

Myhill-Nerode Theorem: Proof (cont.)

$\Longleftarrow$

- Suppose $\sim_L$ has finitely many equivalence classes. We'll construct a DFA M that accepts L.
- Let $x_1, \ldots, x_n \in \Sigma^*$ be representatives for the finitely many equivalence classes of $\sim_L$.
- The states of M are the equivalence classes $[x_1], \ldots, [x_n]$.
- The transition function δ is defined as follows: For all $a \in \Sigma$, $\delta([x_i], a) = [x_i a]$ (the equivalence class of $x_i a$).
- Hey, why is this a proper definition of δ? (Hint: right invariance).

Myhill-Nerode Theorem: Proof (cont.)

$\Longleftarrow$

- The initial state is $[\varepsilon]$.
- The accept states are $F = \{[x_i] \mid x_i \in L\}$.
- Easy: On input $x \in \Sigma^*$, M ends at state $[x]$ (why?).
- Therefore M accepts x iff $x \in L$ (why?).
- So L is accepted by DFA, hence L is regular. ♠

Myhill-Nerode Theorem

- An example: Constructing DFA for the language $L_1 \subset \{0,1\}^*$ contains all strings where the number of 1s is divisible by 5.
- minimizing the number of states of a DFA accepting a given language L.
- The number of equivalence classes is a lower bound on the number of states of any automata that accepts the language.
- There is an automata whose number of states equals the number of equivalence classes.
- any other automata is a refinement of the equivalence relationship (an equivalence class is a union of states).

Minimizing Automata

Input: An automata M
Output: An automata M', such that $L(M) = L(M')$ and M' has a minimal number of states.

- Define an equivalence relationship R, where $R(q)$ is the equivalence class of state q.
- Start with two equivalence classes F and $Q - F$.
- If there is an equivalence class $R(q)$, such that $q_1, q_2 \in R(q)$, and there is $\sigma \in \Sigma$ such that $R(\delta(q_1, \sigma)) \neq R(\delta(q_2, \sigma))$, then split $R(q)$.
- Claim: The algorithm terminates.
- Let R be the states of M'.

More Closure Properties

- Regular languages are closed under complement.
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts L.
- Then $M' = (Q, \Sigma, \delta, q_0, Q - F)$ is a DFA that accepts $\bar{L} = \Sigma^* \setminus L$.
- NFA ?!
- Regular languages are closed under intersection.
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
- Proof with automata ?!

Division of languages

$$L_1/L_2 = \{x | \exists y \in L_2, xy \in L_1\}$$

Examples:

- $L_1 = (01 \cup 1)^*$ and $L_2 = 00$, $L_1/L_2 = ?$
- $L_1/L_2 = \emptyset$.
- $L_3 = a^*b^*c^*$ and $L_4 = b$, $L_3/L_4 = ?$.
- $L_3/L_4 = a^*b^*$.

Division of languages

Theorem: Regular languages are closed under division with any language.
Proof:

- L_1 is a regular language, so it has a DFA $M = (Q, \Sigma, \delta, q_0, F)$.
- L_2 is an arbitrary language.
- For L_1/L_2 we build $M' = (Q, \Sigma, \delta, q_0, F')$.
- $F' = \{q | \exists y \in L_2, \delta(q, y) \in F\}$.
- F' is well define, but might be hard to compute.

Assignments

An assignment substitutes each letter by a language.
Example: $f(1) = a^*$, $f(0) = b$, $f(101) = a^*ba^*$
Theorem: Regular languages are closed under assignment of regular language.
Proof:

- Let R be the regular expression of L. For assignment:
- $f(r_1 \cup r_2) = f(r_1) \cup f(r_2)$.
- $f(r_1 r_2) = f(r_1) f(r_2)$.
- $f((r_1)^*) = f(r_1)^*$.
- Each $f(\sigma) = R_\sigma$, where R_σ is a regular expression.
- We can replace the basic $f(\sigma)$ by R_σ.
- We found a regular expression $f(R)$ for $f(L)$.

Homomorphism

- Homomorphism is an assignment which replaces a letter by a word.
- Example: $h(1) = aba$, $h(0) = aa$, $h(010) = aa\,aba\,aa$ $L_1 = (01)^*$, $h(L_1) = (aaaba)^*$.
- Inverse homomorphism: $h^{-1}(w) = \{x | h(x) = w\}$, $h^{-1}(L) = \{x | h(x) \in L\}$
- Example: $L_2 = (ab \cup ba)^* a$, $h^{-1}(L_2) = \{1\}$.
- Claim: $h(h^{-1}(L)) \subset L \subset h^{-1}(h(L))$.

Homomorphism 2

Theorem: Regular languages are closed under homomorphism and inverse homomorphism.
Proof:

- Homomorphism: special case of assignment.
- Inverse Homomorphism: Let $M = (Q, \Sigma, \delta, q_0, F)$ the automata for L, and $h : \Delta \to \Sigma^*$.
- Proof idea: for each letter $a \in \Delta$ we advance in M using $h(a)$.
- Formally, we define $M' = (Q, \Delta, \delta', q_0, F)$, where $\delta'(q, a) = \delta(q, h(a))$.

Using Homomorphism

- We know that $L_1 = \{0^n 1^n | n \geq 1\}$ is not regular.
- Show that $L_2 = \{a^n b a^n | n \geq 1\}$ is not regular
- We will prove using homomorphism and inverse homomorphism
- $h_1(a) = a$, $h_1(b) = ba$, $h_1(c) = a$.
- $h_2(a) = 0$, $h_2(b) = 1$, $h_2(c) = 1$.
- $h_2(h_1^{-1}(L_2) \cap a^* b c^*) = L_1$
- $h_1^{-1}(L_2) = (a \cup c)^k b (a \cup c)^{k-1}$
- $h_1^{-1}(L_2) \cap a^* b c^* = a^k b c^{k-1}$
- $h_2(h_1^{-1}(L_2) \cap a^* b c^*) = 0^k 1 1^{k-1} = 0^k 1^k$

Algorithmic Questions for NFAs

Q.: Given an NFA, N, and a string w, is $w \in L(N)$?

Answer: Construct the DFA equivalent to N and run it on w.

Q.: Is $L(N) = \emptyset$?
Answer: This is a reachability question in graphs: Is there a path in the states' graph of N from the start state to some accepting state. There are simple, efficient algorithms for this task.

More Algorithmic Questions for NFAs

Q.: Is $L(N) = \Sigma^*$?

Answer: Check if $\overline{L(N)} = \emptyset$.

Q.: Given N_1 and N_2, is $L(N_1) \subseteq L(N_2)$?

Answer: Check if $\overline{L(N_2)} \cap L(N_1) = \emptyset$.

Q.: Given N_1 and N_2, is $L(N_1) = L(N_2)$?

Answer: Check if $L(N_1) \subseteq L(N_2)$ and $L(N_2) \subseteq L(N_1)$.

In the future, we will see that for stronger models of computations, many of these problems cannot be solved by any algorithm.

Another, More Radical Context Switch

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- Myhill-Nerode theorem and pumping lemma for regular languages.

We now introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
- context-free grammars,
- pumping lemma for context-free languages.

Context-Free Grammars

An example of a context free grammar, G_1:

- $A \to 0A1$
- $A \to B$
- $B \to \#$

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\to$ string.
  The left-hand symbol is a variable
  (usually upper-case).
- A string consists of variables and terminals.
- One variable is the start variable.

Rules for Generating Strings

- Write down the start variable (lhs of top rule).
- Pick a variable written down in current string and a derivation that starts with that variable.
- Replace that variable with right-hand side of that derivation.
- Repeat until no variables remain.
- Return final string (concatenation of terminals).

Process is inherently non deterministic.

Example

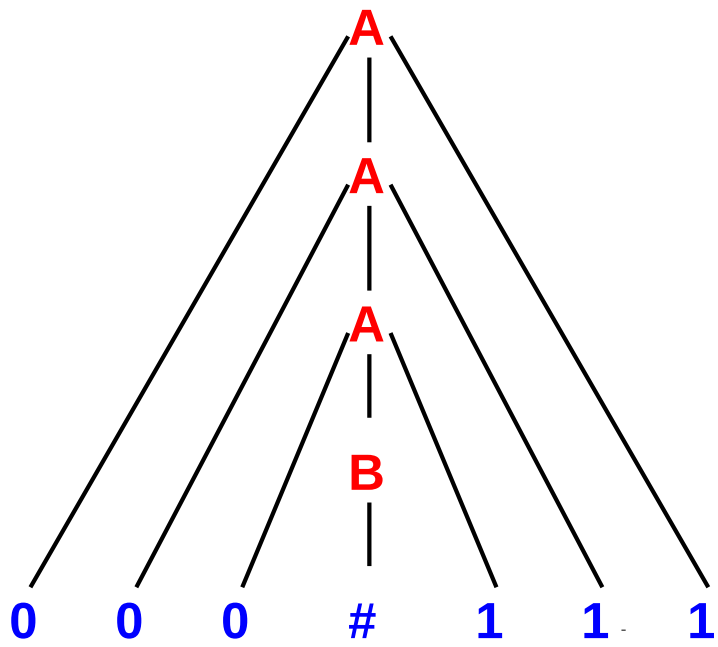
Grammar G_1:

- $A \to 0A1$
- $A \to B$
- $B \to \#$

Derivation with G_1:

$$\begin{aligned} A &\Rightarrow 0A1 \\ &\Rightarrow 00A11 \\ &\Rightarrow 000A111 \\ &\Rightarrow 000B111 \\ &\Rightarrow 000\#111 \end{aligned}$$

A Parse Tree



Question: What strings can be generated in this way from the grammar G_1?
Answer: Exactly those of the form $0^n\#1^n$ $(n \geq 0)$.

Context-Free Languages

The language generated in this way is called the language of the grammar.

For example, $L(G_1)$ is $\{0^n\#1^n | n \geq 0\}$.

Any language generated by a context-free grammar is called a context-free language.

A Useful Abbreviation

Rules with same variable on left hand side

$$\begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \end{aligned}$$

are written as:

$$A \rightarrow 0A1 \mid B$$

English-like Sentences

A grammar G_2 to describe a few English sentences:

$$
\begin{aligned}
< \text{SENTENCE} > &\rightarrow < \text{NOUN-PHRASE} >< \text{VERB} > \\
< \text{NOUN-PHRASE} > &\rightarrow < \text{ARTICLE} >< \text{NOUN} > \\
< \text{NOUN} > &\rightarrow \text{boy} \mid \text{girl} \mid \text{flower} \\
< \text{ARTICLE} > &\rightarrow \text{a} \mid \text{the} \\
< \text{VERB} > &\rightarrow \text{touches} \mid \text{likes} \mid \text{sees}
\end{aligned}
$$

Deriving English-like Sentences

A specific derivation in G_2:

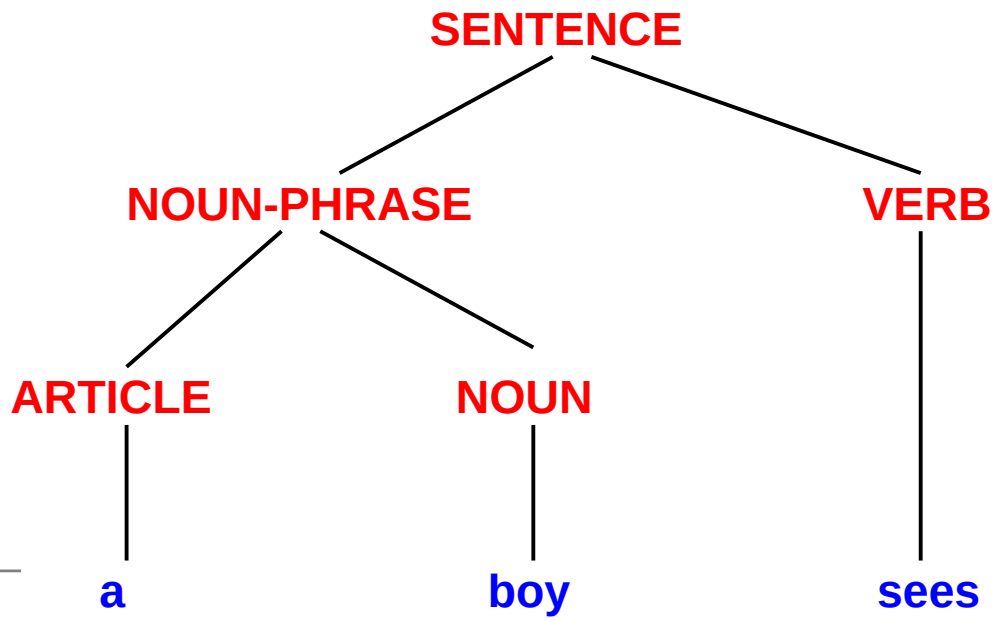
$$
\begin{aligned}
< \text{SENTENCE} > &\Rightarrow < \text{NOUN-PHRASE} >< \text{VERB} > \\
&\Rightarrow < \text{ARTICLE} >< \text{NOUN} >< \text{VERB} > \\
&\Rightarrow \text{a } < \text{NOUN} >< \text{VERB} > \\
&\Rightarrow \text{a boy } < \text{VERB} > \\
&\Rightarrow \text{a boy sees}
\end{aligned}
$$

More strings generated by G_2:

a flower sees
the girl touches

Derivation and Parse Tree

$$
\begin{aligned}
< \text{SENTENCE} > &\Rightarrow < \text{NOUN-PHRASE} >< \text{VERB} > \\
&\Rightarrow < \text{ARTICLE} >< \text{NOUN} >< \text{VERB} > \\
&\Rightarrow \text{a } < \text{NOUN} >< \text{VERB} > \\
&\Rightarrow \text{a boy } < \text{VERB} > \\
&\Rightarrow \text{a boy sees}
\end{aligned}
$$



Formal Definitions

A context-free grammar is a 4-tuple (V, Σ, R, S) where

- V is a finite set of variables,
- Σ is a finite set of terminals,
- R is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
- S is the start symbol.

Formal Definitions

- If u and v are strings of variables and terminals,
- and $A \to w$ is a rule of the grammar, then
- we say uAv yields uwv, written $uAv \Rightarrow uwv$.

We write $u \overset{*}{\Rightarrow} v$ if $u = v$ or

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

for some sequence $u_1, u_2, \ldots, u_k$.

Definition: The language of the grammar is

$$\left\{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \right\} .$$

Example

Consider $G_4 = (V, \{a, b\}, R, S)$.

R (Rules): $S \to aSb \mid SS \mid \varepsilon$.

Some words in the language: $aabb$, $aababb$.

Q.: But what is this language?

Hint: Think of parentheses.

Arithmetic Example

Consider (V, Σ, R, E) where

- $V = \{E, T, F\}$
- $\Sigma = \{a, +, \times, (,)\}$

Rules:

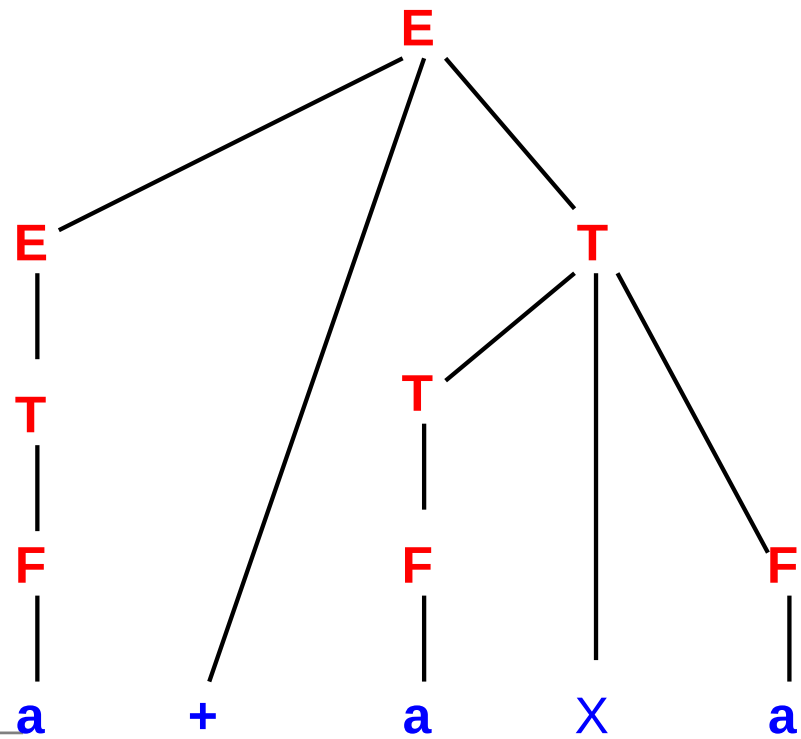
$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T \times F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$

Strings generated by the grammer:
$a + a \times a$ and $(a + a) \times a$.
What is the language of this grammer?

Hint: arithmetic expressions.
$E =$ expression, $T =$ term, $F =$ factor.

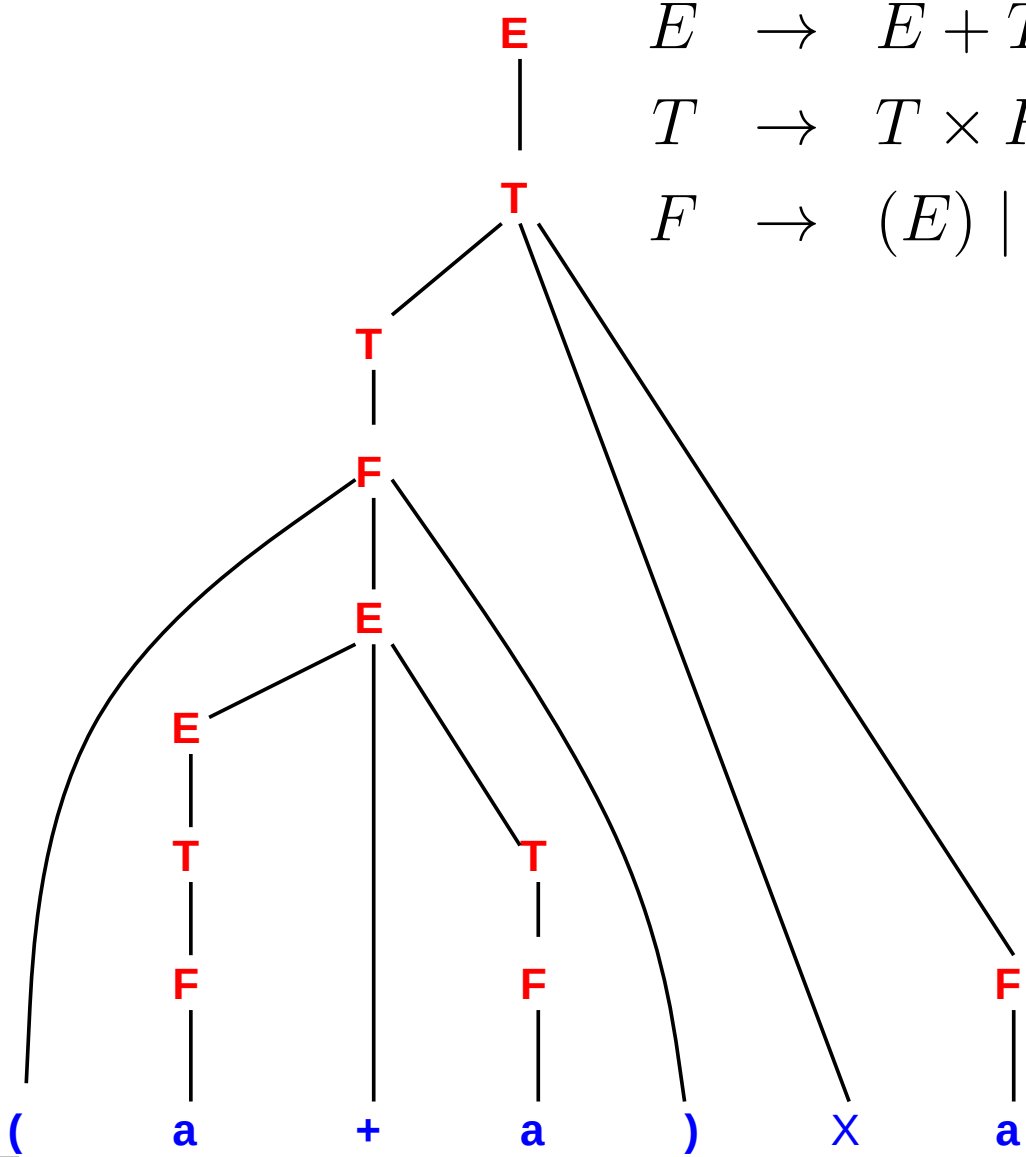
Parse Tree for $a + a \times a$

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T \times F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$



Parse Tree for $(a + a) \times a$

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T \times F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$



Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the union of several CFGs, rename variables (not terminals) so they are disjoint, and add new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_i$.
- To construct CFG for a regular language, "follow" a DFA for the language. For initial state q_0, make R_0 the start variable. For state transition $\delta(q_i, a) = q_j$ add rule $R_i \rightarrow aR_j$ to grammar. For each final state q_f, add rule $R_f \rightarrow \varepsilon$ to grammar.
- For languages with linked substrings (like $\{0^n\#1^n | n \geq 0\}$), a rule of form $R \rightarrow uRv$ may be helpful, forcing desired relation between substrings.

Closure Properties

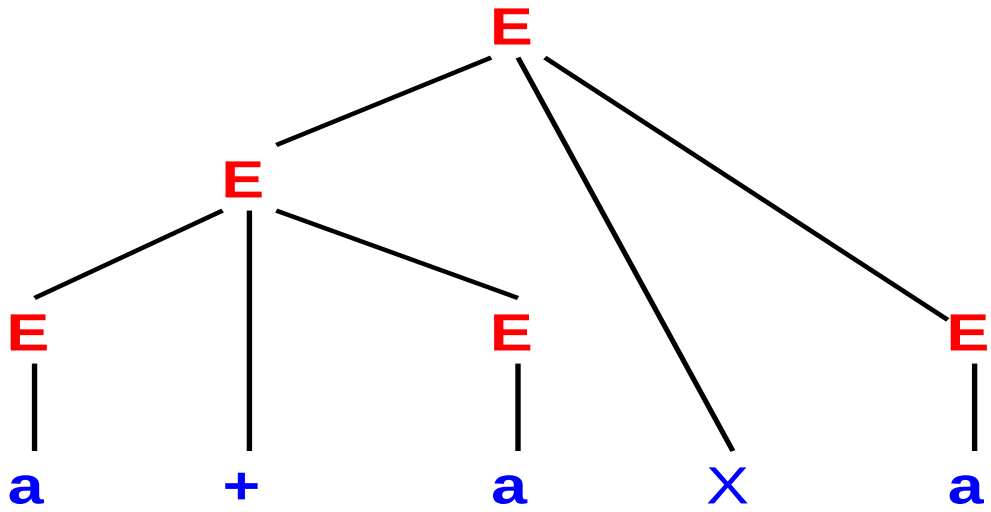
- Regular languages are closed under
  - union
  - concatenation
  - star
- Context-Free Languages are closed under
  - union : $S \rightarrow S_1 \mid S_2$
  - concatenation $S \rightarrow S_1 S_2$
  - star $S \rightarrow \varepsilon \mid SS$

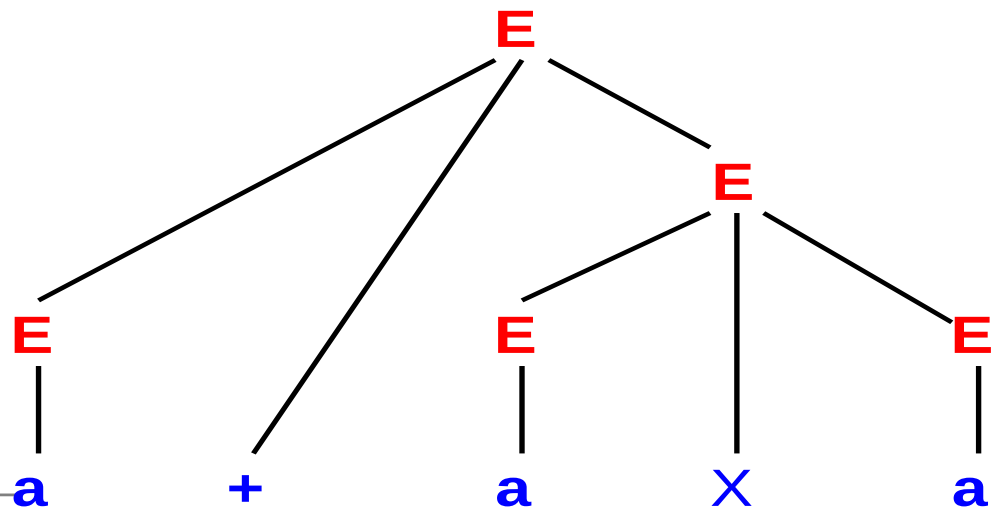
More Closure Properties

- Regular languages are also closed under
  - complement (reverse accept/non-accept states of DFA)
  - intersection $\left(L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \right)$.
- What about complement and intersection of context-free languages?
  - Not clear . . .

Ambiguity

Grammar: $E \rightarrow E{+}E \mid E{\times}E \mid (E) \mid a$





Ambiguity

We say that a string w is derived ambiguously from grammar G if w has two or more parse trees that generate it from G.

Ambiguity is usually not only a syntactic notion but also a semantic one, implying multiple meanings for the same string.

It is sometime possible to eliminate ambiguity by finding a different context free grammar generating the same language. This is true for the grammar above, which can be replaced by the unambiguous grammar from slide 37.

Some languages (*e.g.* $\{1^i 2^j 3^k \mid i = j \text{ or } j = k\}$ are inherently ambiguous.

Chomsky Normal Form

A simplified, canonical form of context free grammers. Every rule has the form

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \\ S &\rightarrow \varepsilon \end{aligned}$$

where S is the start symbol, A, B and C are any variable, except B and C not the start symbol, and A can be the start symbol.

Theorem

Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.
Basic idea:

- Add new start symbol S_0.
- Eliminate all ε rules of the form $A \to \varepsilon$.
- Eliminate all "unit" rules of the form $A \to B$.
- Patch up rules so that grammar generates the same language.
- Convert remaining long rules to proper form.

Proof

Add new start symbol S_0 and rule $S_0 \to S$.
Guarantees that new start symbol does not appear on right-hand-side of a rule.

Proof

Eliminating ε rules.

Repeat:

- remove some $A \to \varepsilon$.
- for each $R \to uAv$, add rule $R \to uv$.
- and so on: for $R \to uAvAw$ add $R \to uvAw$, $R \to uAvw$, and $R \to uvw$.
- for $R \to A$ add $R \to \varepsilon$, except if $R \to \varepsilon$ has already been removed.

until all ε-rules not involving the original start variable have been removed.

Proof

Eliminate unit rules.

Repeat:

- remove some $A \to B$.
- for each $B \to u$, add rule $A \to u$, unless this is previously removed unit rule. (u is a string of variables and terminals.)

until all unit rules have been removed.

Proof

Finally, convert long rules.
To replace each $A \to u_1 u_2 \ldots u_k$ (for $k \geq 3$), introduce new non-terminals

$$N_1, N_2, \ldots, N_{k-1}$$

and rules

$$\begin{aligned} A &\to u_1 N_1 \\ N_1 &\to u_2 N_2 \\ &\vdots \\ N_{k-3} &\to u_{k-2} N_{k-2} \\ N_{k-2} &\to u_{k-1} u_k \end{aligned}$$

♠

Conversion Example

Initial Grammar:

$$\begin{aligned} S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

(1) Add new start state:

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

Conversion Example (2)

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \\
A &\rightarrow B \mid S \\
B &\rightarrow b \mid \varepsilon
\end{aligned}
$$

(2) Remove ε-rule $B \rightarrow \varepsilon$:

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \\
A &\rightarrow B \mid S \mid \varepsilon \\
B &\rightarrow b \mid \varepsilon
\end{aligned}
$$

Conversion Example (3)

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \\
A &\rightarrow B \mid S \mid \varepsilon \\
B &\rightarrow b
\end{aligned}
$$

(3) Remove ε-rule $A \rightarrow \varepsilon$:

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S \\
A &\rightarrow B \mid S \mid \varepsilon \\
B &\rightarrow b
\end{aligned}
$$

Conversion Example (4)

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{aligned}
$$

(4) Remove unit rule $S \rightarrow S$

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{aligned}
$$

Conversion Example (5)

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{aligned}
$$

(5) Remove unit rule $S_0 \rightarrow S$:

$$
\begin{aligned}
S_0 &\rightarrow S \mid ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{aligned}
$$

Conversion Example (6)

$$
\begin{aligned}
S_0 &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{aligned}
$$

(6) Remove unit rule $A \rightarrow B$:

$$
\begin{aligned}
S_0 &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow B \mid S \mid b \\
B &\rightarrow b
\end{aligned}
$$

Conversion Example (7)

$$
\begin{aligned}
S_0 &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow S \mid b \\
B &\rightarrow b
\end{aligned}
$$

Remove unit rule $A \rightarrow S$:

$$
\begin{aligned}
S_0 &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow S \mid b \mid ASA \mid aB \mid a \mid AS \mid SA \\
B &\rightarrow b
\end{aligned}
$$

Conversion Example (8)

$$
\begin{aligned}
S_0 &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \\
A &\rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA \\
B &\rightarrow b
\end{aligned}
$$

(8) Final simplification – treat long rules:

$$
\begin{aligned}
S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow a \\
B &\rightarrow b
\end{aligned}
$$

$\surd$