What we have done so far

- DFAs and regular languages
- NFAs and their equivalence to DFAs
- Regular expressions.
- Regular expressions capture exactly regular languages:
  - Construct a NFA from a regular expression.
  - Left to do: Construct a regular expression from an NFA

Computational Models - Lecture 3

- Constructing a regular expression from an NFA
- What can't DFAs do? Non Regular Languages: Two Approaches
  1. Pumping Lemma
  2. Myhill-Nerode Theorem
- Algorithmic questions for NFAs
- Context Free Grammars (time permitting)

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

Regular Expression from an NFA ($\Longleftarrow$)

We now define generalized non-deterministic finite automata (GNFA).
An NFA:

- Each transition labeled with a symbol or ε,
- reads zero or one symbols,
- takes matching transition, if any.

A GNFA:

- Each transition labeled with a regular expression,
- reads zero or more symbols,
- takes transition whose regular expression matches string, if any.

GNFAs are natural generalization of NFAs.

A Special Form of GNFA

- Start state has outgoing arrows to every other state, but no incoming arrows.
- Unique accept state has incoming arrows from every other state, but no outgoing arrows.
- Except for start and accept states, an arrow goes from every state to every other state, including itself.

Easy to transform any GNFA into special form.

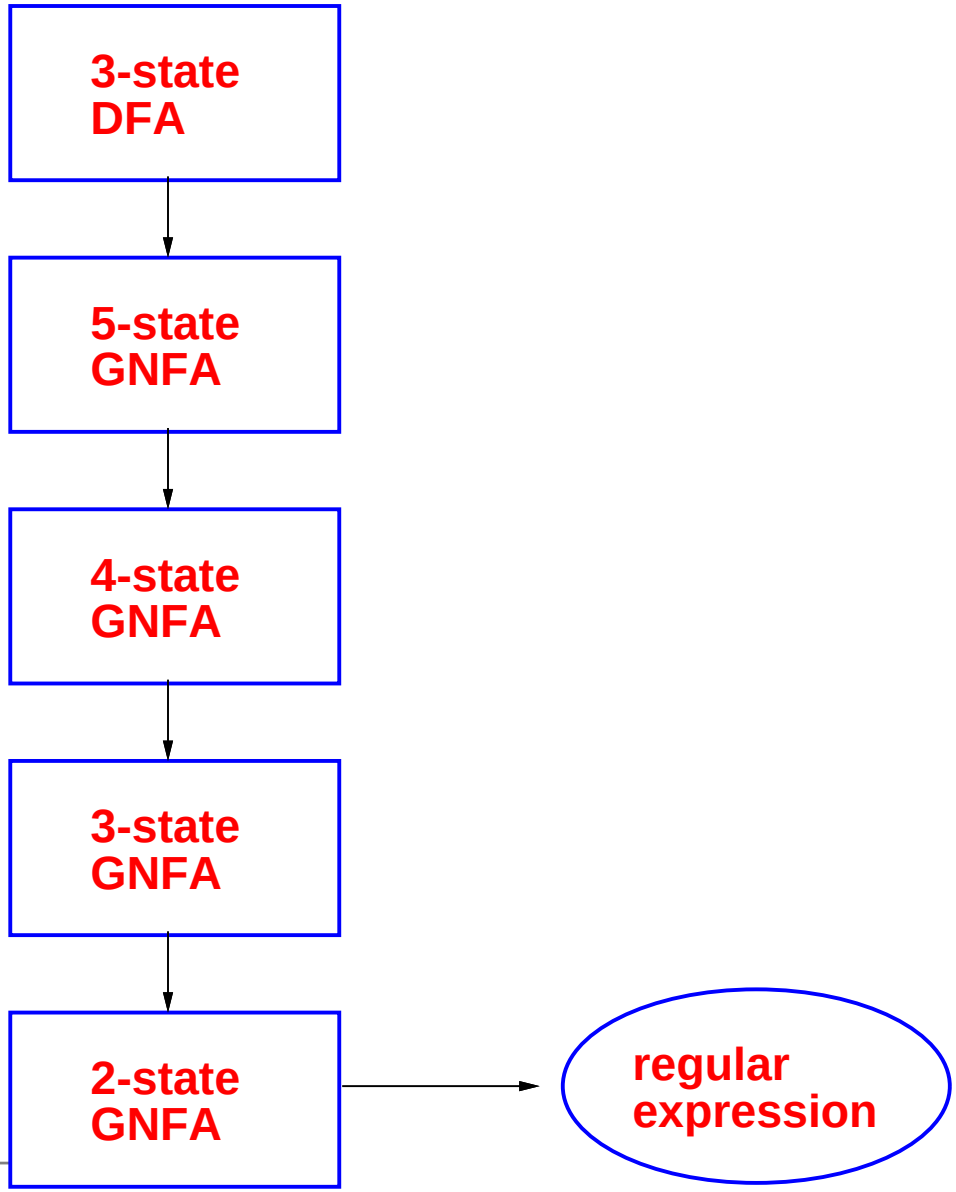
Really? How? . . .

Converting DFA to Regular Expression ($\Longleftarrow$)

Strategy – sequence of equivalent transformations

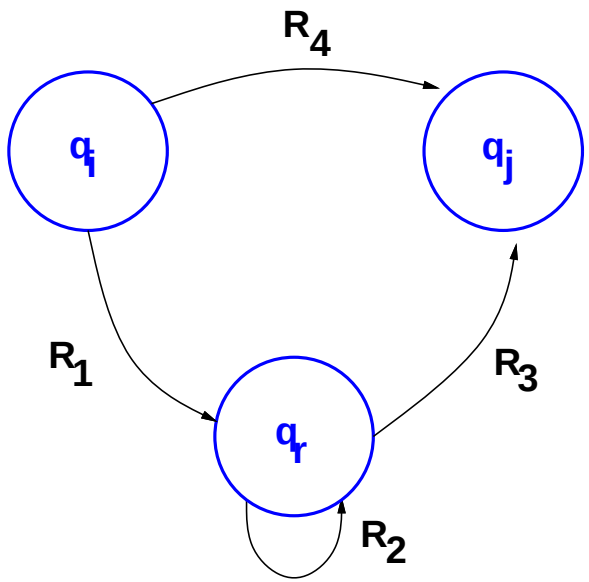
- given a k-state DFA
- transform into $(k+2)$-state GNFA
- while GNFA has more than 2 states, transform it into equivalent GNFA with one fewer state
- eventually reach 2-state GNFA (states are just start and accept).
- label on single transition is the desired regular expression.

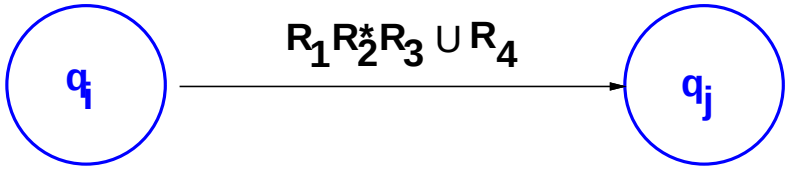
Converting Strategy ($\Longleftarrow$)



Removing One State

We remove one state q_r, and then repair the machine by altering regular expression of other transitions.





Formal Treatment – GNFA Definition

- q_s is start state.
- q_a is accept state.
- $\mathcal{R}$ is collection of regular expressions over Σ.

The transition function is

$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$

If $\delta(q_i, q_j) = R$, then arrow from q_i to q_j has label R.

Arrows connect every state to every other state except:

- no arrow from q_a
- no arrow to q_s

Formal Definition

A generalized deterministic finite automaton (GNFA) is $(Q, \Sigma, \delta, q_s, q_a)$, where

- Q is a finite set of states,
- Σ is the alphabet,
- $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \to \mathcal{R}$ is the transition function.
- $q_s \in Q$ is the start state, and
- $q_a \in Q$ is the unique accept state.

A Formal Model of GNFA Computation

A GNFA accepts a string $w \in \Sigma^*$ if there exists a *parsing* of w, $w = w_1 w_2 \cdots w_k$, where each $w_i \in \Sigma^*$, and there exists a sequence of states $q_0, \ldots, q_k$ such that

- $q_0 = q_s$, the start state,
- $q_k = q_a$, the accept state, and
- for each i, $w_i \in \mathcal{L}(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.
  - (namely w_i is an element of the language described by the regular expression R_i.)

The CONVERT Algorithm

Given GNFA G, convert it to equivalent GNFA G'.

- let k be the number of states of G.
- If $k = 2$, return the regular expression labeling the only arrow.
- If $k > 2$, select any q_r distinct from q_s and q_a.
- Let $Q' = Q - \{q_r\}$.
- For any $q_i \in Q' - \{q_a\}$ and $q_j \in Q' - \{q_s\}$, let
  - $R_1 = \delta(q_i, q_r)$, $R_2 = \delta(q_r, q_r)$,
  - $R_3 = \delta(q_r, q_j)$, and $R_4 = \delta(q_i, q_j)$.
- Define $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$.
- Denote the resulting $k - 1$ states GNFA by G'.

The CONVERT Procedure

We define the recursive procedure CONVERT$(\cdot)$:

Given GNFA G.

- Let k be the number of states of G.
- If $k = 2$, return the regular expression labeling the only arrow of G.
- If $k > 2$, let G' be the $k - 1$ states GNFA produced by the algorithm.
- Return CONVERT(G').

Correctness Proof of Construction

Theorem: G and CONVERT(G) accept the same language.

Proof: By induction on number of states of G

Basis: When there are only 2 states, there is a single label, which characterizes the strings accepted by G.

Induction Step: Assume claim for $k-1$ states, prove for k.

Let G' be the $k-1$ states GNFA produced from G by the algorithm.

G and G' accept the same language

By the induction hypothesis, G' and CONVERT(G') accept the same language.

On input G, the procedure returns CONVERT(G').

So to complete the proof, it suffices to show that G and G' accept the same language.

Three steps:

1. If G accepts the string w, then so does G'.
2. If G' accepts the string w, then so does G.
3. Therefore G and G' are equivalent.

Step One

Claim: If G accepts w, then so does G':

- If G accepts w, then there exists a "path of states" $q_s, q_1, q_2, \ldots, q_a$ traversed by G on w, leading to the accept state q_a.
- If q_r does not appear on path, then G' clearly accepts w (recall that each new regular expression on each edge of G' contains the old regular expression in the "union part").
- If q_r does appear, consider the regular expression corresponding to $\ldots q_i, q_r, \ldots, q_r, q_j \ldots$.
  The new regular expression $(R_{i,r})(R_{r,r})^*(R_{r,j})$ linking q_i and q_j encompasses any such string.
- In both cases, the claim holds.

Steps Two and Three

Claim: If G' accepts w, then so does G.

Proof: Each transition from q_i to q_j in G' corresponds to a transition in G, either directly or through q_r. Thus if G' accepts w, then so does G.

- This completes the proof of the claim that $L(G) = L(G')$.
- Combined with the induction hypothesis, this shows that G and the regular expression CONVERT(G) accept the same language.
- This, in turn, proves our *remarkable claim*:
  A language, L, is described by a regular expression, R, if and only if L is regular. ♣

What We Just Completed

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 quite some progress understanding what finite automata can do. But what can't they do?
Is there a DFA over $\{0, 1\}$ 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 over $\{0, 1\}$ that accepts

- $B = \{0^n 1^n | n \geq 0\}$
- $C = \{w | w$ has an equal number of 0's and 1's$\}$
- $D = \{w | w$ is binary and 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: This is sound intuition. But is this a proof?
Answer: No. D is regular!???
(See problem 7 in hw1...)

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.

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 $\ell = |Q|$, the number of states of M.

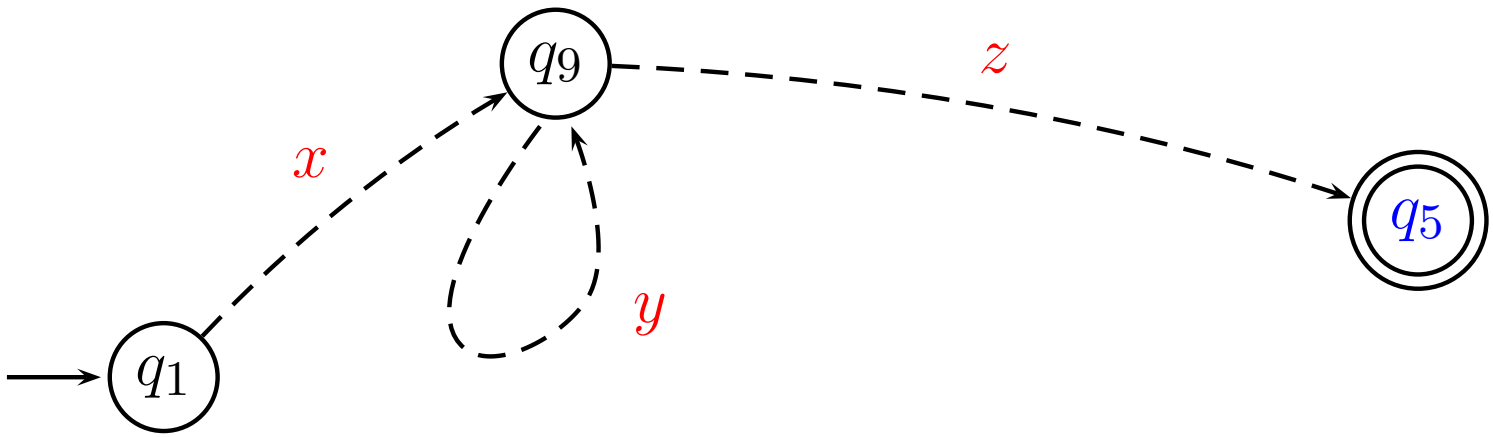
If $s \in L$ has length at least ℓ, consider the sequence of states M goes through as it reads s:

$$
\begin{array}{ccccccccc}
 & s_1 & & s_2 & & s_3 & & s_4 & \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
q_1 & & q_{20} & & q_9 & & q_{17} & & q_{12}
\end{array}
\quad
\begin{array}{ccccccc}
 & s_5 & & s_6 & & \ldots & & s_n & \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
q_{13} & & q_9 & & q_2 & & & & q_5 \in F
\end{array}
$$

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 (this is a DFA, no ϵ-transitions...).
- 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 ℓ 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. ♣

Another Application - first attempt

Theorem: The language
$C = \{w \in \{0,1\}^* | w$ 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.

- The string to be pumped needs to be chosen carefully!
- Let's try the string $s = (01)^\ell$.
- By pumping lemma $s = xyz$, where $xy^kz \in C$ for every k.
- But unfortunately we cannot reach a contradiction, as the chosen string can be pumped as follows:

$$x = \epsilon,\ y = 01,\ z = (01)^{\ell-1}$$

- Then indeed $xy^iz \in L(M)$ for every $i \geq 0$...

Another Application - second attempt

Theorem: The language
$C = \{w \in \{0,1\}^* | w$ 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.
- 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 since $|xy| \leq \ell$, y must be all 0's, contradiction. ♣

Yet Another Application

Theorem: The language $L_3 \subset \{1\}^*$, which contains all strings whose length is a perfect square, is not regular.

$$L_3 = \{1^{n^2} \mid n \geq 0\}$$

Proof:

- Assume for contradiction that L_3 is regular. Let ℓ be the pumping length.
- Take the string $s = 1^{\ell^2}$. By the pumping lemma, $s = xyz$, where for any $i \geq 0$, $xy^iz \in L_3$.
- Note the growing gap between successive members of the sequence of perfect squares:

$$0, 1, 4, 9, 16, 25, 36, 49, ...,$$

Yet Another Application - cont.

- We notice: large members of the sequence cannot be near each other!
- Now consider the strings xy^iz and $xy^{i+1}z$. Their length differs only in the length of y. If we choose a large enough i, these two strings cannot be both in L_3, as they are too close!
- If $m = n^2$, then the difference between it and the next perfect square is:

$$(n+1)^2 - n^2 = n^2 + 2n + 1 - n^2 = 2n + 1 = 2\sqrt{m} + 1$$

- By the pumping lemma, both strings xy^iz and $xy^{i+1}z$ are perfect squares. But if we take $|xy^iz| = m$, it means that they cannot be both perfect squares if $|y| < 2\sqrt{|xy^iz|} + 1$.

Yet Another Application - cont.

We can now calculate the value for i that leads to a contradiction:

- Recall that $|y| \leq |s| = l^2$
- Take $i = l^4$, then:

$$|y| \leq l^2 = \sqrt{l^4} < 2\sqrt{l^4} + 1 \leq 2\sqrt{|xy^i z|} + 1$$

- Thus $xy^{l^4}z$ and $xy^{l^4+1}z$ cannot be both perfect squares - a contradiction!

And now: an alternative technique for showing a language is not regular.

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.

Example: $L = (ab \cup ba)^*$

$\sim_L$ has the following four equivalence classes:

- $[\epsilon] = L$
- $[a] = La$
- $[b] = Lb$
- $[aa] = L(aa \cup bb)\Sigma^*$

Classes of $\sim_L$: More 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 $L_2 = \mathcal{L}(0^*10^*)$. Then $\sim_{L_2}$ has finitely many equivalence classes.
- Let $L_3 \subset \{0,1\}^*$ contain all strings of the form 0^n1^n. Then $\sim_{L_3}$ has infinitely many equivalence classes.

Myhill-Nerode Theorem

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

L is regular $\Longleftrightarrow$ $\sim_L$ 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.
- $L_2 = \mathcal{L}(0^*10^*)$ is regular.
- $L_3 \subset \{0,1\}^*$ contains all strings of the form 0^n1^n. Then L_2 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 $\delta(q_0, x) \in Q$ 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 $\delta(q_0, x) = \delta(q_0, y)$.
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 $x \sim_M y \Longrightarrow x \sim_L y$.
- This means that the equivalence classes of $\sim_M$ refine those of $\sim_L$.
- Thus, the number of equivalence classes of $\sim_M$ is greater or equal than the number of equivalence classes of $\sim_L$.
- Therefore, $\sim_L$ has finitely many equivalence classes, as desired. ♠

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 $x_i a$).
- Why is δ well-defined?
  (Hint: right invariance: if $x \sim_L y$, then $xz \sim_L yz$).

Myhill-Nerode Theorem: Proof (conc.)

$\Longleftarrow$

- 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. ♠

Context Switch

Algorithmic Questions for NFAs

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

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:

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

Terminology:

- Each line is a substitution rule or production.
- Each rule has the form: symbol $\rightarrow$ 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

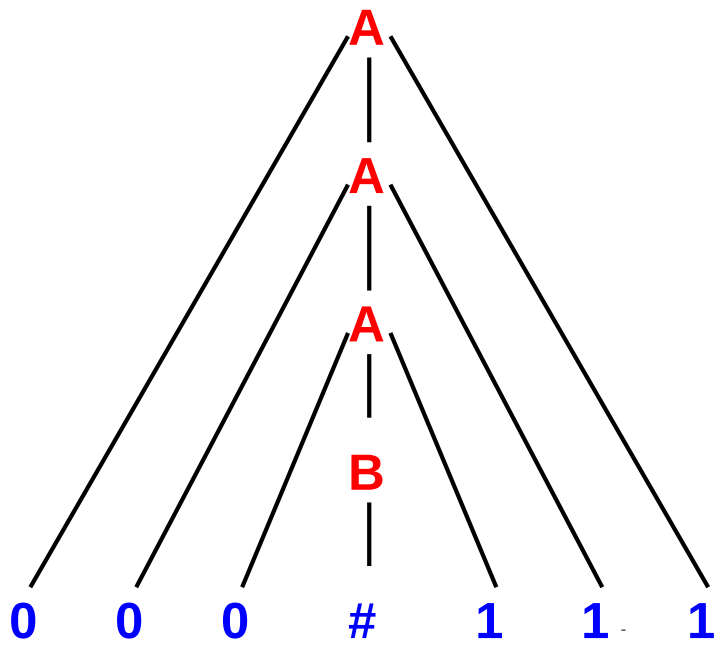
Grammar G_1:

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

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:

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

More strings generated by G_2:

a flower sees
the girl touches

Derivation and Parse Tree

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