Theory of Computation

Finite Automata

Informal Introduction to Finite Automata

What is a Finite Automaton

Remembers only a finite amount of information.
Information represented by its state.
State changes in response to inputs.
Rules that tell how the state changes in response to inputs are called transitions.

Language of an Automaton

The set of strings accepted by an automaton A is the language of A.
Denoted $L(A)$.

Central Concepts of Automata Theory

Alphabets

An alphabet is any finite set of symbols.

Strings

A string over an alphabet Σ is a list, each element of which is a member of Σ.
Σ^* is set of all strings over alphabet Σ.
ε stands for the empty string.

Languages

A language is a subset of Σ^* for some alphabet Σ.

Problems

A problem is the question of deciding whether a given string is a member of some particular language.

Deterministic Finite Automata (DFA)

Deterministic Finite Automata

A DFA Consists of

1. A finite set of states (Q).
2. An input alphabet (Σ).
3. A transition function (δ)
4. A start state (q_0, in Q).

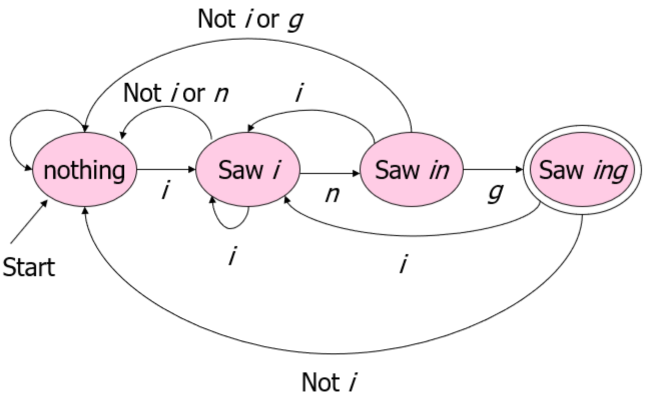
5. A set of final/accepting states ($F \subseteq Q$).

The Transition Function

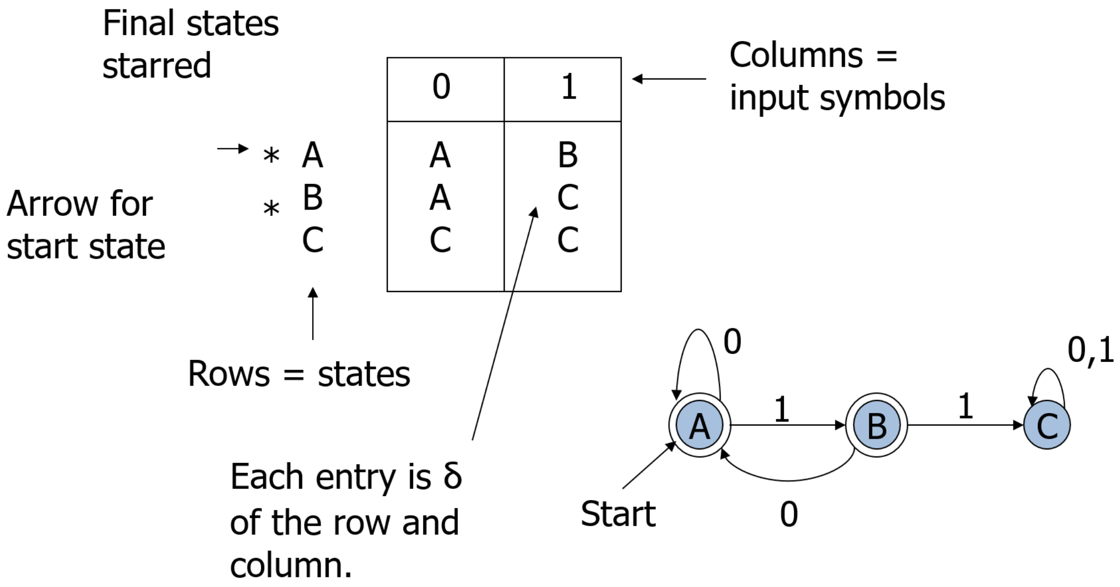
$\delta(q, a) =$ the state that the DFA foes to when it is in state q and input a is received.
Note: always a next state - add a dead state if no transition.

Graph Representation of DFA

Nodes = states.
Arcs represent transition function.
Arrow labeled "Start" to the start state.
Final states indicated by double circles.
For example (Recognizing Strings Ending in "ing"):



Alternative Representation: Transition Table



Final states starred

Columns = input symbols

	0	1
→ ∗ A	A	B
∗ B	A	C
C	C	C

Arrow for start state

Rows = states

Each entry is δ of the row and column.

Inductive Definition of Extented δ

Basis: $\delta(q, \varepsilon) = q$
Induction: $\delta(q, wa) = \delta(\delta(q, w), a)$

Language of a DFA

For a DFA A, $L(A)$ is the set of strings labeling paths from the start state to a final state.
Formally: $L(A) =$ the set of strings w such that $\delta(q_0, w)$ is in F.

Regular Languages

A language L is regular if it is the language accepted by some DFA.
Note: the DFA must accept only the strings in L, no others.

Nondeterministic Finite Automata(NFA)

Nondeterminism

A nondeterministic finite automaton has the ability to be in several states at once.
Start in one start state, accept if any sequence of choices leads to a final state.

Formal NFA

1. A finite set of states (Q)
2. An input alphabet (Σ)
3. A transition function (δ)
4. A start state in Q (q_0)
5. A set of final states $F \subseteq Q$

$\delta(q, a)$ is a set of states.
Basis: $\delta(q, \varepsilon) = \{q\}$
Induction: $\delta(q, wa) =$ the union over all states p in $\delta(q, w)$ of $\delta(p, a)$

Language of an NFA

The language of the NFA is the set of strings it accepts.

Equivalence of DFA and NFA

A DFA can be turned into an NFA that accepts the same language.
If $\delta_D(q, a) = p$, let the NFA have $\delta_N(q, a) = \{p\}$
For any NFA there is a DFA that accepts the same language.
Proof is the subset construction.
Thus, NFA's accept exactly the regular languages.

Subset Construction

Given an NFA with states Q, inputs Σ, transition function δ_N, start state q_0, and final states F, construct equivalent DFA with:

1. States 2^Q (Set of subsets of Q)
2. Inputs Σ
3. Start state $\{q_0\}$
4. Final states = all those with a member of F

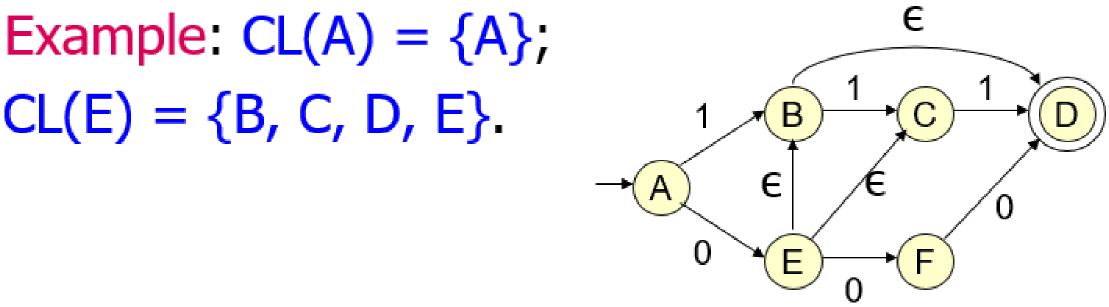
The transition function δ_D is defined by: $\delta_D(\{q_1, \ldots, q_k\}, a)$ is the union over all $i = 1, \ldots, k$ of $\delta_N(q_i, a)$.

NFA with ε-Transition

We can allow state-to-state transitions on ε input.
A convenience at times, but still only regular languages are accpted.

Closure of States

$CL(q) =$ set of states you can reach from state q following only arcs labeled ε.



Example: CL(A) = {A};

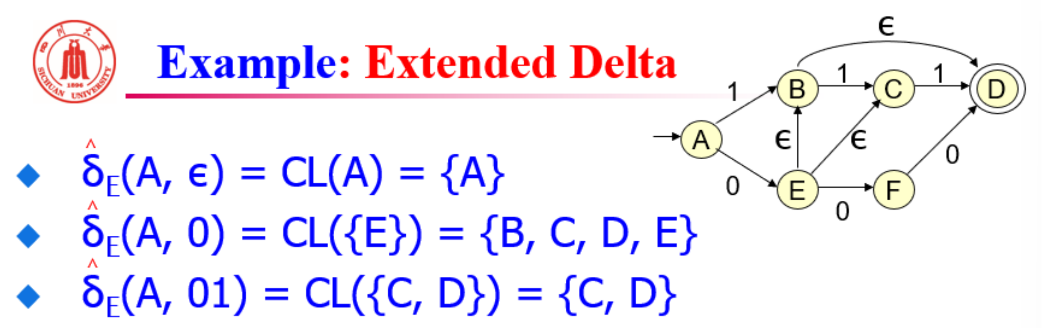
CL(E) = {B, C, D, E}.

Closure of a set of states = union of the closure of each state.

Extended Delta for ε-NFA

Intuition: $\hat{\delta}_E(q, w)$ is the set of states you can reach from q following a path labeled w.
Basis: $\hat{\delta}_E(q, \varepsilon) = CL(q)$.
Induction: $\hat{\delta}_E(q, xa)$ is computed by:

1. Start with $\hat{\delta}_E(q, x) = S$.
2. Take the union of $CL(\delta_E(p, a))$ for all p in S.



Example: Extended Delta

- $\hat{\delta}_E(A, \epsilon) = CL(A) = \{A\}$
- $\hat{\delta}_E(A, 0) = CL(\{E\}) = \{B, C, D, E\}$
- $\hat{\delta}_E(A, 01) = CL(\{C, D\}) = \{C, D\}$

Language of an ε-NFA is the set of strings w such that $\hat{\delta}_E(q_0, w)$ contains a final state.

Equivalence of NFA, ε-NFA

Start with an ε-NFA with states Q, inputs Σ, start state q_0, final states F, and transition function δ_E.
Construct an "ordinary" NFA with states Q, inputs Σ, start state q_0, final states F', and transition function δ_N.
Compute $\delta_N(q, a)$ as follows:

1. Let $S = CL(q)$.
2. $\delta_N(q, a)$ is the union over all p in S of $\delta_E(p, a)$.

$F' =$ the set of states q such that $CL(q)$ contains a state of F.

Regular Language

Regular Expressions

Introduction to RE

Regular Expressions describe languages by an algebra.
If E is a regular expression, then $L(E)$ is the language it defines.

Operations on Languages

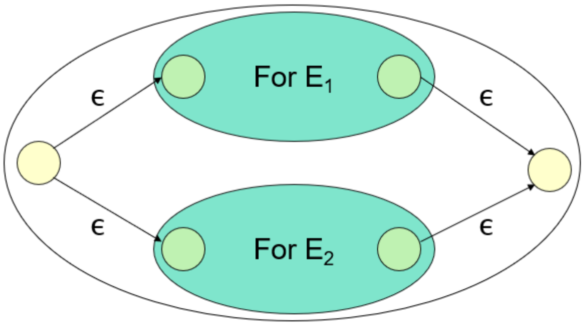
RE's use three operations: union, concatenation, and Kleene star.
Union: The union of sets.
Concatenation: The concatenation of languages L and M is denoted LM. It contains every string wx such that w is in L and x is in M.
Kleene Star: $L^* = \{\varepsilon\} \bigcup L \bigcup LL \bigcup LLL \bigcup \dots$

Definition of RE

Basis 1: If a is any symbol, then a is a RE, and $L(a) = \{a\}$.
Basis 2: ε is a RE, and $L(\varepsilon) = \{\varepsilon\}$.
Basis 3: $\varnothing$ is a RE, and $L(\varnothing) = \varnothing$.
Induction 1: If E_1 and E_2 are regular expressions, then $E_1 + E_2$ is a regular expression, and $L(E_1 + E_2) = L(E_1) \bigcup L(E_2)$.
Induction 2: If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$.
Induction 3: If E is a RE, then E^* is a RE, and $L(E^*) = (L(E))^*$.

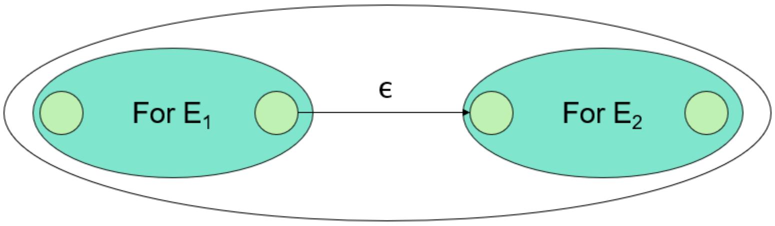
RE to $\varepsilon-$NFA

Union:



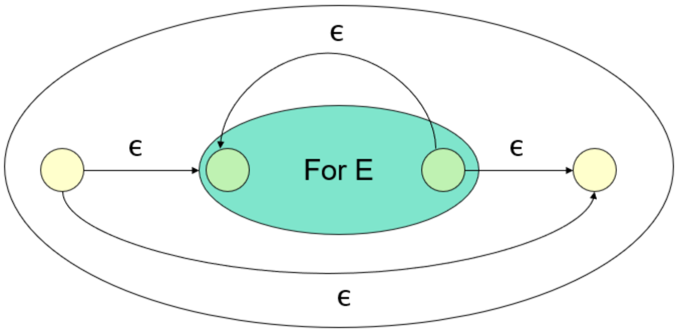
For $E_1 \cup E_2$

Concatenation:



For E_1E_2

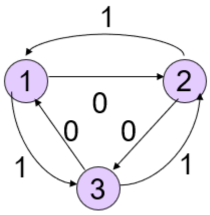
Closure:



For E*

DFA to RE

A strange sort of induction. States of the DFA are named $1, 2, \ldots, n$. Induction is on k, the maximum state number we are allowed to traverse along a path.
A k-path is a path through the graph of the DFA that goes through no intermediate state numbered higher than k.
n-paths are unrestricted. RE is the union of RE's for the n-paths from the start state to each final state.
Example:



0-paths from 2 to 3:
RE for labels = **0**.

1-paths from 2 to 3:
RE for labels = **0+11**.

2-paths from 2 to 3:
RE for labels =
(10)*0+1(01)*1

3-paths from 2 to 3:
RE for labels = ??

Let ${R_{ij}}^k$ be the regular expression for the set of labels of k-paths from state i to state j.
A k-path from i to j either:

1. Never goes through state k
2. Goes through k one or more times

${R_{ij}}^k = {R_{ij}}^{k-1} + {R_{ik}}^{k-1}({R_{kk}}^{k-1})^* {R_{kj}}^{k-1}$
The RE with the same language as the DFA is the sum (union) of ${R_{ij}}^n$, where:

1. n is the number of states; i.e., paths are unconstrained.
2. i is the start state.
3. j is one of the final states.

Algebraic Laws for RE's

$+$ is commutative and associative; concatenation is associative.

Identities and Annihilators

$\varnothing$ is the identity for $+$. $R + \varnothing = R$
ε is the identity for concatenation. $\varepsilon R = R\varepsilon = R$.
$\varnothing$ is the annihilator for concatenation. $\varnothing R = R\varnothing = \varnothing$.

Decision Properties of Regular Languages

Properties of Language Classes

A language class is a set of languages. Language classes have two important kinds of preperties:

1. Decision properties.
2. Closure properties.

Closure Properties

A closure property of a language class says that given languages in the class, an operation produces another language in the same class.

Decision Properties

A decision property for a class of languages is an algorithm that takes a formal description of a language and tells whether or not some property holds.

The Emptiness Problem

Given a regular language, does the language contain any string at all?
Assume representation is DFA. Compute the set of states reachable from the start state. If at least one final state is reachable, then yes, else no.

The Infiniteness Problem

Is a given regular language infinite? Start with a DFA for the language.
Key Idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.

Proof of Key Idea

If an n-state DFA accepts a string w of length n or more, then there must be a state that appears twice on the path labeled w from the start state to a final state. Because there are at least $n + 1$ states along the path.

Second Key Idea

There is a string of length $> n$ ($=$ number of states) in L, if and only if there is a string of length between n and $2n - 1$.

Completion of Infiniteness Algorithm

Test for membership all strings of length between n and $2n - 1$. If any are accepted, then infinite, else finite.
Better: find cycles between the start state and a final state.

Finding Cycles

1. Eliminate states not reachable from the start state.
2. Eliminate states that do not reach a final state.
3. Test if the remaining transition graph has any cycles.

A simple, less efficient way to find cycles is to search forward from a given node N. If you can reach N, then there is a cycle. Do this starting at each node.

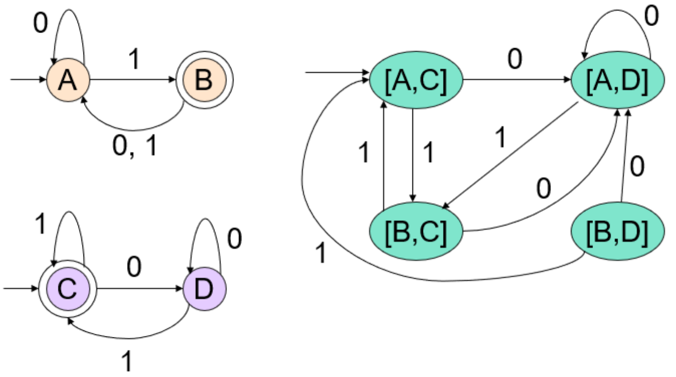
The Pumping Lemma

For every regular language L. There is an integer n, such that for every string w in L of length $\geq n$ we can write $w = xyz$ such that:

1. $|xy| \leq n$.
2. $|y| > 0$.
3. For all $i > 0$, $xy^i z$ is in L.

Decision Property: Equivalence

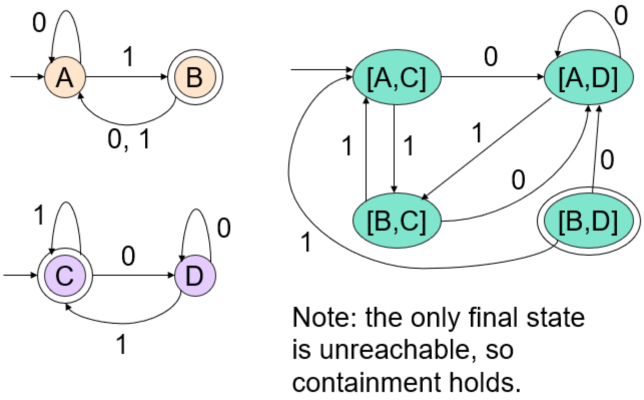
Given regular languages L and M, is $L = M$?
Algorithm involves constructing the product DFA from DFA's for L and M.
Let these DFA's have sets of states Q and R, respectively. Product DFA has set of states $Q \times R$.
That is, pairs $[q, r]$ with q in Q, r in R.
Start state $= [q_0, r_0]$ (the start states of the DFA's for L, M).
Transitions: $\delta([q, r], a) = [\delta_L(q, a), \delta_M(r, a)]$. That is, we simulate the two DFA's in the two state components of the product DFA.
Example:



Make the final states of the product DFA be those states $[q, r]$ such that exactly one of q and r is a final state of its own DFA. Thus, the product accepts w iff w is in exactly one of L and M. $L = M$ if and only if the product automaton's language is empty.

Decision Property: Containment

Given regular languages L and M, is $L \subseteq M$?
How do you define the final states $[q, r]$ of the product so its language is empty iff $L \subseteq M$?
Answer: q is final; r is not.
In the example,



Note: the only final state is unreachable, so containment holds.

Only $[B, D]$ is final. $[B, D]$ is not reachable from the start state. Thus, the language of the product automaton is empty, and we conclude that the language of the orange automaton is a subset of the language of the purple automaton.

The Minimum-State DFA for a Regular Language

In principle, since we can test for equivalence of DFA's we can, given a DFA A find the DFA with the fewest states accepting $L(A)$.
A terrible algorithm: Test all smaller DFA's for equivalence with A.
An efficient algorithm: Construct a table with all pairs of states. If you find a string that distinguishes two states, mark that pair. Algorithm is a recursion on the length of the shortest distinguishing string.

Transitivity of "Indistinguishable"

If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.

Constructing the Minimum-State DFA

Suppose $q_1, \ldots, q_k$ are indistinguishable states. Replace them by one representative state q.
Then $\delta(q_1, a), \ldots, \delta(q_k, a)$ are all indistinguishable states. Let $\delta(q, a) =$ the representative state for that group.

Closure Properties of Regular Languages

Closure Under Union, Concatenation and Kleene Closue

If L and M are regular languages, so is $L \bigcup M$, LM, L^*.

Closure Under Intersection

If L and M are regular languages, then so is $L \bigcap M$.

Closure Under Difference

If L and M are regular languages, then so is $L - M =$ strings in L but not M.

Closure Under Complementation

The complement of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$.

Closure Under Reversal

Given language L, L^R is the set of strings whose reversal in in L.
If E is $F + G$, then $E^R = F^R + G^R$.
If E is FG, then $E^R = G^R F^R$.
If E is F^*, then $E^R = (F^R)^*$.

Homomorphism

A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet.

Closure under Homomorphism

If L is a regular language, and h is a homomorphism on its alphabet, then
$h(L) = \{h(w)|w\ is\ in\ L\}$ is also a regular language.

Inverse Homomorphism

Let h be a homomorphism and L a language whose alphabet is the output language of h.
$h^{-1}(L) = \{w|h(w)\ is\ in\ L\}$.

Context-Free Grammar

Formal Definition

Informal Comments

A context-free grammar is a notation for describing languages.

CFG Formalism

Terminals $=$ symbols of the alphabet of the language being defined.
Variables $=$ nonterminals $=$ a finite set of other symbols, each of which represents a language.
Start symbol $=$ the variable whose language is the one being defined.

Productions

A production has the form variable (head) $\rightarrow$ string of variables and terminals (body).
Example: Formal CFG
Here is a formal CFG for $\{0^n1^n|n \geq 1\}$.
Terminals $= \{0, 1\}$.
Variables $= \{S\}$.
Start symbol $= S$.
Productions $=$
$S \rightarrow 01$
$S \rightarrow 0S1$

Derivations

Derivations - Formalism

We say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \rightarrow \gamma$ is production.

Iterated Derivation

$\Rightarrow^*$ means "zero or more derivation steps."

Sentential Forms

α is a sentential form iff $S \Rightarrow^* \alpha$.

Language of a Grammar

If G is a CFG, then $L(G)$, the language of G, is $\{w|S \Rightarrow^* w\}$.

Context-Free Languages

A language that is defined by some CFG is called a context-free language.

Backus-Naur Form

BNF Notation

Variables are words in $< \dots >$;
Example: $< statement >$.
Terminals are often multicharacter strings indicated by boldface or underline;
Example: **while** or $\underline{WHILE}$.
Symbol $::=$ is often used for $\rightarrow$.
Symbol $|$ is used for "or."
Example: $S \rightarrow 0S1|01$ is shorthand for $S \rightarrow 0S1$ and $S \rightarrow 01$.

Kleene Closure:
Symbol $\dots$ is used for "one or more."
Example: $< digit >::= 0|1|2|3|4|5|6|7|8|9$
$< unsigned\ integer >::=< digit > \dots$
Translation: Replace $\alpha \dots$ with a new variable A and productions $A \rightarrow A\alpha|\alpha$.
Example: Kleene Closure
Grammar for unsigned integers can be replaced by:
$U \rightarrow UD|D$
$D \rightarrow 0|1|2|3|4|5|6|7|8|9$

Optional Elements:
Surround one or more symbols by $[\dots]$ to make them optional.
Example:
$< statement >::= if < condition > then < statement > [; else < statement >]$
Translation: replace $[\alpha]$ by a new variable A with productions $A \rightarrow \alpha|\varepsilon$.

Grouping:
Use $\{\dots\}$ to surround a sequence of symbols that need to be treated as a unit.
Example: $< statement\ list >::=< statement > [\{; < statement >\} \dots]$

Left- and Rightmost Derivations

Leftmost Derivations

Say $wA\alpha \Rightarrow_{lm} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
Also, $\alpha \Rightarrow^*_{lm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{lm}$ steps.

Rightmost Derivations

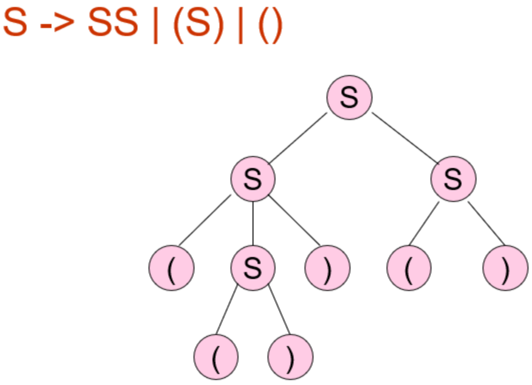
Say $\alpha Aw \Rightarrow_{rm} \alpha\beta w$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
Also, $\alpha \Rightarrow^*_{rm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{rm}$ steps.

Parse Trees

Definitions

Parse Trees

Parse trees are trees labeled by symbols of a particular CFG.
Leaves: labeled by a terminal or ε.
Interior nodes: labeled by a variable.
Root: must be labeled by the start symbol.
Example:



Yield of a Parse Tree

The concatenation of the labels of the leaves in left-to-right order is called the yield of the parse tree.
Example: the yield of the parse tree above is $(())()$.

Relationship to Left- and Rightmost Derivations

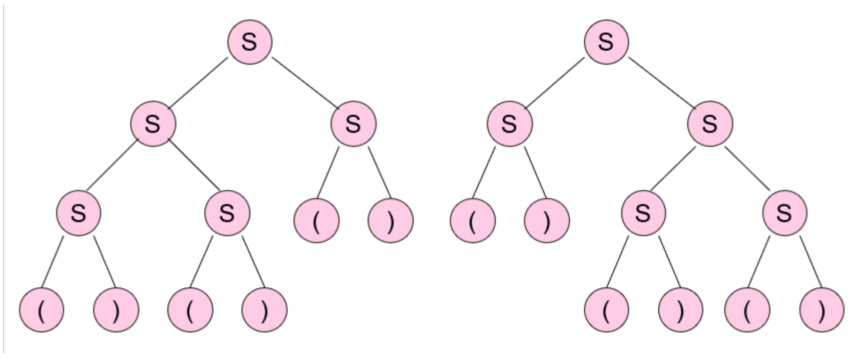
Parse Trees, Leftmost and Rightmost Derivations

If there is a parse tree with root labeled A and yield w, then $A \Rightarrow^*_{lm} w$.
If $A \Rightarrow^*_{lm} w$, then there is a parse tree with root A and yield w.

Ambiguity in Grammars

Ambiguous Grammars

A CFG is ambiguous if there is a string in the language that is the yield of two or more parse trees.
Example:



Equivalent definitions of "ambiguous grammar" are:

1. There is a string in the language that has two different leftmost derivations.
2. There is a string in the language that has two different rightmost derivations.

Inherent Ambiguity

Certain CFL's are inherently ambiguous, meaning that every grammar for the language is ambiguous.

Normal Form for CFG

Eliminating Useless Variables

Variables that Derive Nothing

Consider: $S \to AB$, $A \to aA|a$, $B \to AB$.
S derives nothing, and the language is empty.

Algorithms to Eliminate Variables that Derive Nothing

1. Discover all variables that derive terminal strings.
2. For all other variables, remove all productions in which they appear in either the head or body.

Unreachable Symbols

Algorithm: Remove from the grammar all symbols not discovered reachable from S and all productions that involve these symbols.

Eliminating Useless Symbols

A symbol is useful if it appears in some derivation of some terminal string from the start symbol. Otherwise, it is useless.
Eliminate all useless symbols by:

1. Eliminate symbols that derive no terminal string.

2. Eliminate unreachable symbols.

Removing Epsilon

Epsilon Productions

We can almost avoid using productions of the form $A \to \varepsilon$ (called $\varepsilon-$productions).
Theorem: If L is a CFL, then $L - \{\varepsilon\}$ has a CFG with no $\varepsilon-$productions.

Nullable Symbols

To eliminate $\varepsilon-$productions, we first need to discover the nullable symbols $=$ variables A such that $A \Rightarrow^* \varepsilon$.
Example: Nullable Symbols
$S \to AB$, $A \to aA|\varepsilon$, $B \to bB|A$.
Basis: A is nullable because of $A \to \varepsilon$.
Induction: B is nullable because of $B \to A$.
Then, S is nullable because of $S \to AB$.

Eliminating $\varepsilon-$Productions

$S \to AB$, $A \to aA|\varepsilon$, $B \to bB|A$.
Key idea: turn each production $A \to X_1 \dots X_n$ into a family of productions.
For each subset of nullable X's, there is one production with those eliminated from the right side "in advance".
Example:
$S \to ABC$, $A \to aA|\varepsilon$, $B \to bB|\varepsilon$, $C \to \varepsilon$
Note: C is now useless.
New grammar:
$S \to AB|A|B$
$A \to aA|a$
$B \to bB|b$

For all variables A:

1. If $w \neq \varepsilon$ and $A \Rightarrow^*_{old} w$, then $A \Rightarrow^*_{new} w$.
2. If $A \Rightarrow^*_{new} w$ then $w \neq \varepsilon$ and $A \Rightarrow^*_{old} w$.

Removing Unit Productions

Cleaning Up a Grammar

Theorem: if L is a CFL, then there is a CFG for $L - \{\epsilon\}$ that has:

1. No useless symbols.
2. No $\epsilon-$productions.
3. No unit productions.

Chomsky Normal Form

Chomsky Normal Form

A CFG is said to be in Chomsky Normal Form if every production is of one of these two forms:

1. $A \to BC$ (body is two variables)
2. $A \to a$ (body is a single terminal)

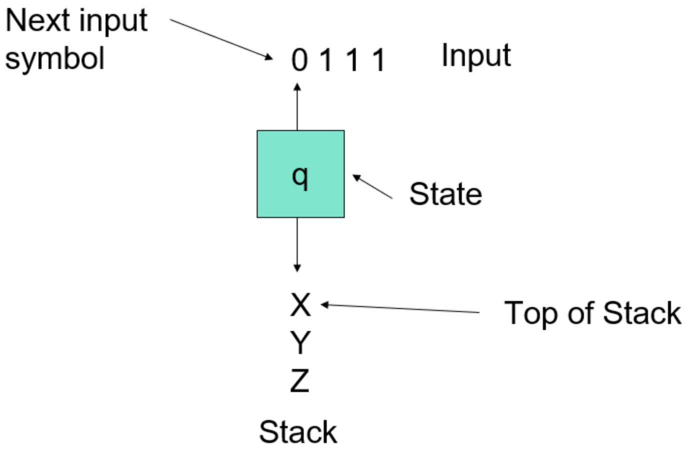
Theorem: If L is a CFL, then $L - \{\epsilon\}$ has a CFG in CNF.

Pushdown Automata

Definition

Pushdown Automata

The PDA is an automaton equivalent to the CFG in language-defining power.



PDA Formalism

A PDA is described by:

1. A finite set of states (Q)
2. An input alphabet (Σ)
3. A stack alphabet (Γ)
4. A transition function (δ)
5. A start state (q_0)
6. A start symbol (Z_0)
7. A set of final states ($F \subseteq Q$)

Moves of the PDA

The Transition Function

Take three arguments:

1. A state, in Q.
2. An input, which is either a symbol in Σ or ϵ.
3. A stack symbol in Γ.

$\delta(q, a, Z)$ is a set of zero or more actions of the form (p, α).

Actions of PDA

If $\delta(q, a, Z)$ contains (p, α) among its actions, then one thing the PDA can do in state q, with a at the front of the input, and Z on top of the stack is:

1. Change the state to p.
2. Remove a from the front of the input (but a may be ϵ).
3. Replace Z on the top of the stack by α.

Instantaneous Descriptions

An instantaneous description (ID) is a triple (q, w, α), where:

1. q is the current state.
2. w is the remaining input.
3. α is the stack contents, top at the left.

The "Goes-To" Relation

To say the ID I can become ID J in one move of the PDA, we write $I \vdash J$.
Formally, $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$ for any w and α, if $\delta(q, a, X)$ contains (p, β).
Extend $\vdash$ to $\vdash^*$, meaning "zero or more moves".

Languages of the PDA

Language of a PDA

The common way to define the language of a PDA is by final state.
If P is a PDA, then $L(P)$ is the set of string w such that $(q_0, w, Z_0) \vdash^* (f, \epsilon, \alpha)$ for final state f and any α.
Another language defined by the same PDA is by empty stack.
If P is a PDA, then $N(P)$ is the set of strings w such that $(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)$ for any state q.

Equivalence of Language Definitions

1. If $L = L(P)$, then there is another PDA P' such that $L = N(P')$.
2. If $L = N(P)$, then there is another PDA P'' such that $L = L(P'')$.

Deterministic PDA's

Deterministic PDA

To be deterministic, there must be at most one choice of move for any state q, input symbol a, and stack symbol X. In addition, $\delta(q, a, X)$ and $\delta(q, \epsilon, X)$ cannot both be nonempty.

Equivalence of PDA and CFG

Conversion of CFG to PDA

Converting a CFG to a PDA

Let $L = L(G)$.
Construct PDA P such that $N(P) = L$.
P has:
One state q.
Input symbols $=$ terminals of G.
Stack symbols $=$ all symbols of G.
Start symbol $=$ start symbol of G.

Intuition About P

At each step, P represents some left-sentential form (step of a leftmost derivation).
If the stack of P is α, and P has so far consumed x from its input, then P represents left-sentential form $x\alpha$.
At empty stack, the input consumed is a string in $L(G)$.

Transition Function of P

1. $\delta(q, a, a) = (q, \epsilon)$.
2. If $A \to \alpha$ is a production of G, then $\delta(q, \epsilon, A)$ contains (q, α).

Conversion of PDA to CFG

From a PDA to a CFG

Now, assume $L = N(P)$. Construct a CFG G such that $L = L(G)$.
Intuition: G will have variables $[pXq]$ generating exactly the inputs that cause P to have the net effect of popping stack symbol X while going from state p to state q.

Variable of G

G's variables are of the form $[pXq]$.
This variable generates all and only the strings w such that $(p, w, X) \vdash^* (q, \epsilon, \epsilon)$.

Productions of G

Each production for $[pXq]$ comes from a move of P in state p with stack symbol X.
Simplest case: $\delta(p, a, X)$ contains (q, ϵ). Then the production is $[pXq] \to a$
Next simplest case: $\delta(p, a, X)$ contains (r, Y) for some state r and symbol Y. G has production $[pXq] \to a[rYq]$.
Third simplest case: $\delta(p, a, X)$ contains (r, YZ) for some state r and symbols Y and Z. Now, P has replaced X by YZ.
General case: Suppose $\delta(p, a, X)$ contains $(r, Y_1, \ldots Y_k)$ for some state r and $k \geq 3$.
Generate family of productions $[pXq] \to a[rY_1s_1][s_1Y_2s_2]\ldots[s_{k-2}Y_{k-1}s_{k-1}][s_{k-1}Y_kq]$
Add to G another variable S, the start symbol, and add productions $S \to [q_0Z_0p]$ for each state p.

Pumping Lemma for CFL

Statement

Statement of the CFL Pumping Lemma

For every context-free language L. There is an integer n, such that for every string z in L of length $\geq n$ there exists $z = uvwxy$ such that:

1. $|vwx| \leq n$.
2. $|vx| > 0$
3. For all $i \geq 0$, uv^iwx^iy is in L.

Properties of CFL

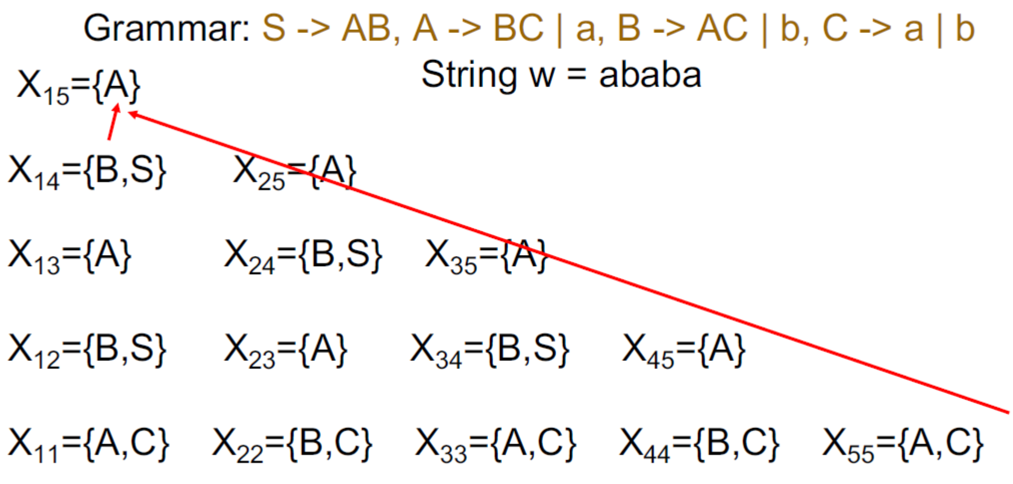
Decision Properties

Testing Emptiness

Eliminate useless variables. If the start symbol is one of these, then CFL is empty; otherwise not.

Testing Membership

Want to know if string w is in $L(G)$.
Algorithm (CYK) is a good example of dynamic programming and runs in time $O(n^3)$, where $n = |w|$.
Example of CYK algorithm:



Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa

X_{15}={A}				
X_{14}={B,S}	X_{25}={A}			
X_{13}={A}	X_{24}={B,S}	X_{35}={A}		
X_{12}={B,S}	X_{23}={A}	X_{34}={B,S}	X_{45}={A}	
X_{11}={A,C}	X_{22}={B,C}	X_{33}={A,C}	X_{44}={B,C}	X_{55}={A,C}

Testing Infiniteness

Use the pumping lemma constant n. If there is a string in the language of length between n and $2n - 1$, then the language is infinite; otherwise not.

Closure Properties

Closure of CFL

Closed: union, concatenation, Kleene closure, reversal, homomorphisms and inverse homomorphisms.
Not closed: intersection and difference.

Closure

Union

Form a new grammar for $L \bigcup M$ by combining all the symbols and productions of G and H. Then, add a new start symbol S. Add productions $S \rightarrow S_1|S_2$.

Concatenation

Form a new grammar for LM by starting with all symbols and productions of G and H. Add a new start symbol S. Add production $S \rightarrow S_1S_2$.

Star

Let L have grammar G, with a start symbol S_1. Form a new grammar for L^* by introducing to G a new start symbol S and the productions $S \rightarrow S_1S|\epsilon$.

Reversal

If L is a CFL with grammar G, form a grammar for L^R by reversing the body of every production.
Example: Let G have $S \rightarrow 0S1|01$. The reversal of $L(G)$ has grammar $S \rightarrow 1S0|10$.

Homomorphism

Let L be a CFL with grammar G. Let h be a homomorphism on the terminal symbols of G.
Construct a grammar for $h(L)$ by replacing each terminal symbol a by $h(a)$.
Example: G has productions $S \rightarrow 0S1|01$. h is defined by $h(0) = ab$, $h(1) = \epsilon$. $h(L(G))$ has the grammar with productions $S \rightarrow abS|ab$.

Inverse Homomorphism

Let $L = L(P)$ for some PDA P. Construct PDA P' to accept $h^{-1}(L)$.
Formal Constructions of P' :
States are pairs $[q, w]$, where:

1. q is a state of P.
2. w is a suffix of $h(a)$ for some symbol a.

Stack symbols of P' are those of P.
Start state of P' is $[q_0, \epsilon]$.
Input symbols of P' are the symbols to which h applies.
Final states of P' are the states $[q, \epsilon]$ such that q is a final state of P.

Nonclosure

Intersection

Example:
$L_1 = \{0^n 1^n 2^n | n \geq 1\}$ is not a CFL. $L_2 = \{0^n 1^n 2^i | n \geq 1, i \geq 1\}$ and $L_3 = \{0^i 1^n 2^n | n \geq 1, i \geq 1\}$ are. But $L_1 = L_2 \bigcap L_3$.

Difference

More general : Any class of languages that is closed under difference is closed under intersection.

Turing Machine

Finite