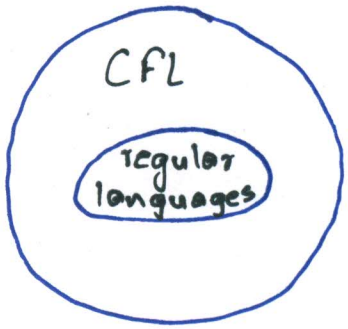
Week 12: Lecture Notes

Relationship of the regular and context-free languages

- All regular languages are CFL (as every FA can be trivially considered as a PDA)



- If L is NOT a CFL, then it cannot be regular. (Pumping lemma can be used to show that L is not CFL)
- CFL- is closed under
  a. Union b. Concatenation and
  c. Kleen's star
  - trivial language ϕ and $\{a\}$ are definitely CFLs
    - ϕ generated with no rules
    - $\{a\}$ generated with only one rule $S \to a$
- So the class of context-free languages must contain all regular languages, the closure of the trivial languages under these operations.

Direct Construction

- $M = (Q, \Sigma, \delta, q_0, f)$ – a DFA
- CFG: $G = (V = \{Q \cup \Sigma\}, T = \Sigma, P, S = q_0)$, P consists of

$$P = \{ q \to ap \mid \delta(q,a) = p \} \cup \{ q \to \varepsilon \mid q \in F \}$$

- $L(M) = L(G)$

Properties of CFL

The (BAR-HILLEL) Pumping lemma for CFLs.

Lemma:

Let L be any CFL. Then there is a constant n, depending only on L, such that if z is in L and $|z| \geq n$, then we may write $z = uvwxy$ such that

i. $|vx| \geq 1$

ii. $|vwx| \leq n$, and

iii. $\forall i \geq 0$, uv^iwx^iy is in L

Proof:

Let G be a Chomsky Normal Form (CNF) grammar generating $L - \{\varepsilon\}$

(No grammar can be found if $L = \phi$ or $\{\varepsilon\}$

- However, if $L = \phi$, then the statement of the theorem is still valid as $\exists$ no such string z in ϕ
- CNF grammar G will actually generate $L - \{\varepsilon\}$, which is again not of importance, as we shall surely pick $n > 0$, in which case z cannot be ε any way.)

Observe that if $z \in L(G)$ and z is long, then any parse tree for must contain a long path.

More precisely we have the following claim.

Claim:

Suppose, we have a parse tree according to a CNF grammar $G = (V, T, P, S)$, and suppose that the yield of the tree is a terminal string w.

If the length of the longest path is n, then

$$|w| \leq 2^{n-1}$$

Proof: We prove it by induction on n.

Base: $n = 1$ — trivial

S — variable

|

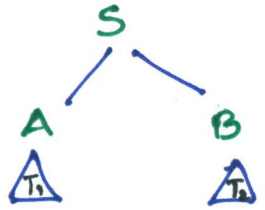
a — terminal

$$|a| = 1 = 2^{n-1} = 2^0$$

Induction: Suppose the longest path has length n, and $n > 1$

As $n > 1$, we cannot start the tree using a production with a terminal.

Apply induction on subtrees rooted at A and B which have no path of length $> n-1$



$\Rightarrow$ yields of subtree rooted at A as well as B have lengths $\leq 2^{n-2}$

$$|w_1| \leq 2^{n-2}, \quad |w_2| \leq 2^{n-2}$$

$\Rightarrow$ yield of the entire tree rooted at S has size

$$\leq 2^{n-2} + 2^{n-2} = 2^{n-1}$$

Hence the proof of the claim.

- Let G have K variables and let $n = 2^K$

Let z is in $L(G)$ and $|z| \geq n$.

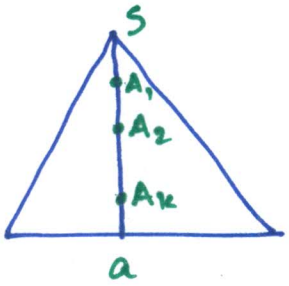
Now by the claim, any parse tree whose longest path is of length K or less must have a yield of length $2^{K-1} = n/2$ or less.

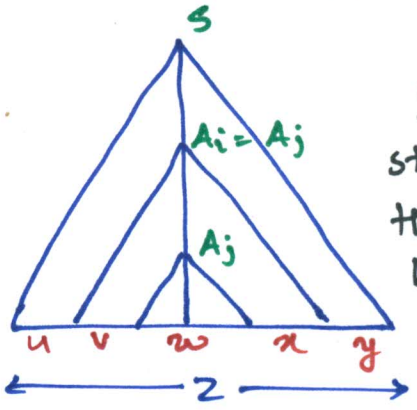
Such a parse tree cannot have yield z, because z is too long, $|z| \geq n$

Thus any parse tree with yield z has a path of length at least $K+1$

But such a path has atleast $K+2$ vertices, all but the last are labeled by variables

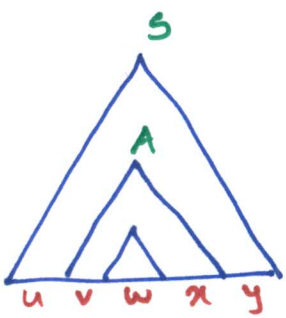
$\Rightarrow$ There must be some vertex that appears twice on the path.

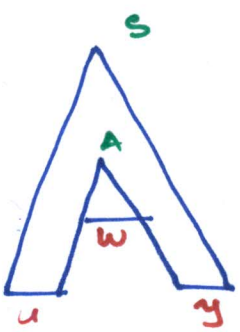


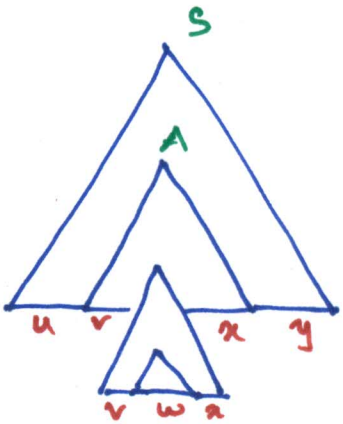


Dividing the string w so that it can be pumped

Every suffieiently long string in L must have a long path in it's parse tree







We can say more

Some variables must appear twice near the bottom of the path. In particular, P be a path that is longer than or as long as any path in the tree.

Then there must be two vertices v_1 and v_2 on the path satisfying the following conditions:

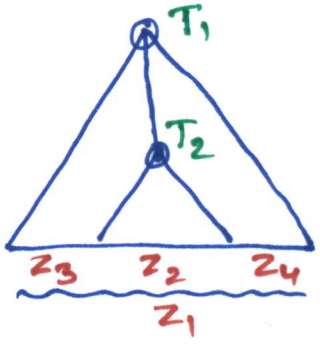
i. The vertices v_1 and v_2 both have the same label, say A

ii. Vertex v_1 is closer to the root than vertex v_2

iii. The portion of the path from v_1 to the leaf is of length at most $K+1$ (i.e. $\leq K+1$)

- To see that v_1 and v_2 can always be found, just proceed up path P from the leaf, keeping track of the labels encountered.
- Of the first $K+2$ vertices, only the leaf has a terminal label.
- The remaining $K+1$ vertices cannot have distinct variable labels
- Subtree T_1 rooted at v_1 represents the derivation of a subword of length $\leq 2^K$, as longest path in T_1 is of length $K+1$, since P was a path of longest length in the entire tree.
- Let z_1 be the yield of subtree T_2
- If T_2 is rooted at v_2 and z_2 is the yield of T_2, then

$z_1 = z_3 z_2 z_4$



- Furthermore, z_3, z_4 cannot be both ε as the first derivation of z_1 must be of the form $A \to BC$, for some variables B, C
- The subtree T_2 must be completely within either the subtree generated by B or the subtree generated by C

We now know that

$$A \underset{G}{\overset{*}{\Rightarrow}} z_3 A z_4 \text{ and } A \underset{G}{\overset{*}{\Rightarrow}} z_2, \text{ where } |z_2 z_3 z_4| \leq 2^k = n$$

But it follows that

$$A \underset{G}{\overset{*}{\Rightarrow}} z_3^i z_2 z_4^i \text{ for each } i \geq 0$$

Thus the string z can clearly be written as

$$z = u z_3 z_2 z_4 y \text{ for some } u, y$$

We let $z_3 = v$, $z_2 = w$, $z_4 = x$ to complete the proof.

Applications of the Pumping for CFL's

(Proof of non-context-freeness of a language)

Example: $L = \{0^n 1^n 2^n \mid n \geq 1\}$ is not a CFL.

Let L be CFL and n be the constant of the pumping lemma for CFLs

Consider $z = 0^n 1^n 2^n$

Write $z = uvwxy$, where $|vwx| \leq n$ and $vx \neq \varepsilon$

Note that vwx cannot involve both 0's and 2's, ·

Since the last 0 and the first 2 are separated by $n+1$ positions:

1. vwx has no 2's - Then vx consists of only 0's and/and has atleast one of these symbols. Then uwy which would have to be in L by

the PL, has n 2's, but fewer than n 0's or fewer than n 1's or both. It therefore doesnot belong in L.

2. vwx has no 0's - similarly uwy has n 0's, but fewer 1's or fewer 2's or both. It therefore cannot be in L

We get contradiction in both, hence L is not a CFL.

Example: $L = \{x \in \{0,1\}^* \mid |x| \text{ is a perfect square}\}$

Let L be a CFL and n be the constant of the PL for the CFLs

Consider $z = 0^{n^2}$

Write $z = uvwxy$ with $|vwx| \leq n$ and $vx \neq \varepsilon$

Let $|vx| = m$, Then $m \leq n$

Now consider the string uv^2wx^2y.

By the PL for CFLs, $uv^2wx^2y \in L$

But $$n^2 < |uv^2wx^2y| = n^2 + m \leq n^2 + n < (n+1)^2$$

$\Rightarrow |uv^2wx^2y|$ is not a perfect square.

Therefore $uv^2wx^2y \notin L$

Which is a contradiction to the PL for the CFLs.

$\therefore$ L is not a CFL

Example: $L = \{a^i b^j c^i d^j \mid i \geq 1, j \geq 1\}$ is not CFL

Let L be CFL and n be the constant of the PL for CFLs

Consider $z = a^n b^n c^n d^n = uvwxy$ with $|vwx| \leq n$ and

$|vx| \geq 1$ and $uv^i wx^i y \in L \ \forall\ i \geq 0$

Then vx will have at most two different symbol.

If vx has two different symbols, then they are consequtive

i.e. either a, b, or b, c or c, d

- vx has only a's – then uwy has fewer a's than c's

  Similarly, if vx has only b's or only c's or only a's

  $\Rightarrow uwy \notin L$

- vx has only a's and b's – then also uwy has fewer a's, then

  similarly if vx has only b's and c's or c's and d's

  $\Rightarrow uwy \notin L$

Thus L is not CFL.

Example: $L = \{a^p \mid p \text{ is a prime}\}$ is not a CFL

Let L be a CFG and n be the constant of the PL for CFL's

Let $z = a^p \in L$, $p > n$

By PL for CFL, $z = uvwxy$ with $|vwx| \leq n$, $|vx| \geq 1$

and $uv^i wx^i y \in L \ \forall\ i \geq 0$

for $i = 0$, $uv^0 wx^0 y = uwy \in L \Rightarrow |uwy| = q$ must be a prime

Let $|vx| = r$.

Then $|uv^q wx^q y| = |uwy| + q|vx| = q + rq$

$\neq$ a prime

$\Rightarrow uv^q wx^q y \notin L$

Ogden's Lemma

Let L be a CFL. Then there is a constant n (which may infact be the same as for the Pumping Lemma) such that if z is any word in L with $|z| \geq n$, and we mark any n or more positions of z "distinguished", then we can write $z = uvwxy$ such that

i. vx has atleast one distinguished position

ii. vwx has atmost distinguished position

iii. for all $i \geq 0$, $uv^i wx^i y \in L$

- Bar-Hillel Pumping lemma for CFL is a special case of Ogden's Lemma, in which all positions are distinguished.

- There are certain non-CFL's for which Bar-Hillel pumping lemma for CFLs is of no use.

Example:

$L = \{a^i b^j c^k d^l \mid \text{either } i = 0 \text{ or } j = k = l\}$ is not CFL

- Let $z = b^j c^k d^l = uvwxy \in L$

Then it is always possible to choose u, v, w, x, y so that $uv^m wx^m y \in L \ \forall m$

for example, if we choose vwx to have only one b's

- Let $z = a^i b^j c^j d^j = uvwxy$

Then v, x might consist only of a's in which case

$$uv^m wx^m y \in L \ \forall m$$

$\Rightarrow$ PL is of no use

Closure Properties of CFLs

Theorem: CFLs are closed under union, concatenation and Kleene closure

Proof: Let L_1, L_2 be CFLs generated by the CFGs

$G_1 = (V_1, T_1, P_1, S_1)$, $G_2 = (V_2, T_2, P_2, S_2)$.

We assume V_1, V_2 are disjoint (if not we may rename variables at will without changing the language)

Let $S_3, S_4, S_5 \notin V_1 \cup V_2$

For $L_1 \cup L_2$, construct grammar

$$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, P_3, S_3)$$

where P_3 is $P_1 \cup P_2$ plus the production $S_3 \to S_1 | S_2$

- if $w \in L_1$, then $S_3 \underset{G_3}{\Rightarrow} S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ in a derivation in G_3 as every production of G_1 is a production of G_3
- Similarly, every word in L_2 has a derivation in G_3 begining with $S_3 \underset{G_3}{\Rightarrow} S_2$

Thus, $L_1 \cup L_2 \subseteq L(G_3)$ —— (1)

For the converse let $w \in L(G_3)$.

Then $S_3 \overset{*}{\underset{G_3}{\Rightarrow}} w$ begins with either $S_3 \underset{G_3}{\Rightarrow} S_1 \overset{*}{\underset{G_3}{\Rightarrow}} w$

or $S_3 \underset{G_3}{\Rightarrow} S_2 \overset{*}{\underset{G_3}{\Rightarrow}} w$

Since, V_1, V_2 are disjoint, only symbols of G_1 may appear in the derivation $S_1 \overset{*}{\Rightarrow} w$

Also, any production of P_3 that involve only symbols of G_1 are those from P_1

Hence, only productions of G_1 are used in the derivation $S_1 \overset{*}{\underset{G_3}{\Rightarrow}} w$ which in turn implies $w \in L_1$

Analogously, if the derivation starts $S_3 \underset{G_3}{\Rightarrow} S_2$ we may conclude $w \in L_2$

Hence $L(G_3) \subseteq L_1 \cup L_2$ —— (2)

From (1) and (2) $L(G_3) = L_1 \cup L_2$

For concatenation ($L_1 L_2$)

Let $G_4 = (V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, P_4, S_4)$ where P_4 is $P_1 \cup P_2$ plus the production $S_4 \to S_1 S_2$

A proof that $L(G_4) = L(G_1) L(G_2)$ is similar to the proof for union and is omitted.

For closure (L_1^*)

Let $G_5 = (V_1 \cup \{S_5\}, T_1, P_5, S_5)$, where P_5 is P_1 plus the productions $S_5 \to S_1 S_5 \mid \varepsilon$

We again leave the proof that $L(G_5) = L(G_1)^*$ to the readers.

Theorem

The CFLs are closed under multiplication.

Proof:

Let L be a CFL, $L \subseteq \Sigma^*$ and for each $a \in \Sigma$, let L_a be a CFL..

Let $L = L(G)$ and for each $a \in \Sigma$, let $L_a = L(G_a)$

Without loss of generality assume that the variables of G and G_a are disjoint.

Now, construct a grammar G' as follows:

variables of G' : all variables of G and G_a

terminals of G' : terminals of G_a

Start symbol of G': start symbol of G

production of G': all productions of G_a plus productions obtained from a production $A \to \alpha$ of G by substituting S_a in place of each $a \in \Sigma$ appearing in α.

Example: $L = \{w \in \{a,b\}^* \mid w \text{ has equal number of a's and b's}\}$

$L_a = \{0^n 1^n \mid n \geq 1\}$, $L_b = \{ww^R \mid w \text{ is in } (0+2)^*\}$

For G we may choose

$$S \to aSbS \mid bSaS \mid \varepsilon$$

For G_a take

$$S_a \to 0S_a1 \mid 01$$

For G_b take

$$S_b \to 0S_b0 \mid 2S_b2 \mid \varepsilon$$

Now if f is a substitution $f(a) = L_a$, $f(b) = L_b$, then $f(L)$ is generated by the grammar

$$S \to S_aSS_bS \mid S_bSS_aS \mid \varepsilon$$

$$S_a \to 0S_a1 \mid 01$$

$$S_b \to 0S_b0 \mid 2S_b2 \mid \varepsilon$$

Theorem: If L is a CFL over alphabet Σ and f is a substitution on Σ such that $f(a)$ is a CFL for each $a \in \Sigma$ then $f(L)$ is a CFL.

Proof:

if $w = a_1 a_2 \dots a_n \in \Sigma^*$

then $f(w) = \{x_1 x_2 \dots x_n \mid x_i \in f(a_i),\ i = 1,2,\dots n\}$

We write $f(w) = f(a_1) f(a_2) \dots f(a_n)$

CFG G' construction for $f(L)$ from (i) CFG G_a for $f(a)$ $a \in \Sigma$ and (ii) CFG G for L.

- Variables of G' = variables of G + variables of G_a for $a \in \Sigma$
- Terminals of G' = Terminals of G_a for $a \in \Sigma$
- Start symbol of G' = Start symbol of G
- production of G' = productions of G_a + productions of G with each terminal a in their bodies replaced by S_a everywhere a occurs.

To prove $L(G') = f(L)$

Claim: $w \in L(G')$ iff $w \in f(L)$

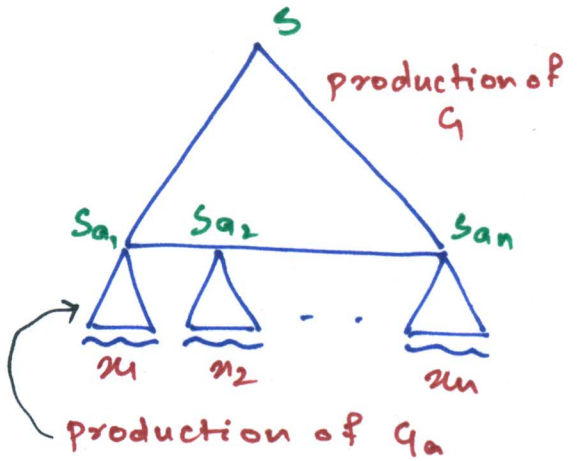
proof: (if)

Let $w \in f(L)$

$\Rightarrow \exists\ x = a_1 a_2 \ldots a_n \in L$

and $x_i \in f(a_i)$, $i = 1, 2 \ldots n$

such that $w = x_1 x_2 \ldots x_n$



- A parse tree in G' begins with a parse tree of G, and finishes with many parse trees, each in one of the grammar of G_a

$$S \overset{*}{\underset{G'}{\Rightarrow}} x_1 x_2 \ldots x_n = w$$

$$S \overset{*}{\underset{G}{\Rightarrow}} S_{a_1} S_{a_2} \ldots S_{a_n} \overset{*}{\underset{G_a}{\Rightarrow}} x_{a_1} S_{a_2} \ldots S_{a_n} \Rightarrow \ldots$$

$$\underset{G_{a_n}}{\Rightarrow} x_{a_1} x_{a_2} \ldots x_{a_n} = w$$

$\therefore w \in L(G')$

(only if)

Let $w \in L(G')$

- if w has a parse tree T it must look like above as variables of G and G_a are disjoint
- Thereby, we can easily identify a string $a_1 a_2 \ldots a_n \in L(G)$ and strings $x_i \in f(a_i)$, $i = 1, 2 \ldots n$ such that
  - i. $w = x_1 x_2 \ldots x_n$
  - ii. The string $S a_1 S a_2 \ldots S a_n$ is the yield of a tree that is formed from T by deleting some subtrees.

But $x_1 x_2 \ldots x_n \in f(L)$ as it is formed by substituting strings x_i for each of the a_i's.

Hence, $w \in f(L)$

$\therefore \quad L(G') = f(L)$

Observation

The closure of CFL's under substitution implies

i. closure under union

ii. closure under concatenation

iii. closure under *

- Union of L_a and L_b is simply the substitution of L_a and L_b into $\{a, b\}$
- $L_a L_b$ → substitution into $\{ab\}$
- L_a^* → substitution into a^*
- Homomorphism is a special type of substition. Hence we have the following:

  The CFL's are closed under homomorphism.

Theorem: If L is CFL, then so is L^R (reversal)

(Hint: Reverse each production of CFG for L

$A \to \alpha$ is in G, $\Rightarrow A \to \alpha^R$ is in G^R)

Theorem:

The CFLs are not closed under intersection

Proof: (By counter example)

$L = \{a^i b^i c^i \mid i \geq 1\}$ is not CFL by PL for CFLs

Now consider $L_1 = \{a^i b^i c^j \mid i \geq 1, j \geq 1$

$L_2 = \{a^i b^j c^j \mid i \geq 1, j \geq 1\}$

Both L_1 and L_2 are CFLs

However $L_1 \cap L_2 = L$ is NOT CFL

Corollary:

The CFLs are not closed under complementation.

Proof: CFLs are closed under union.

If they are closed under complementation, then by De Morgan's law, $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$ would be closed under intersection, which is a contradiction.

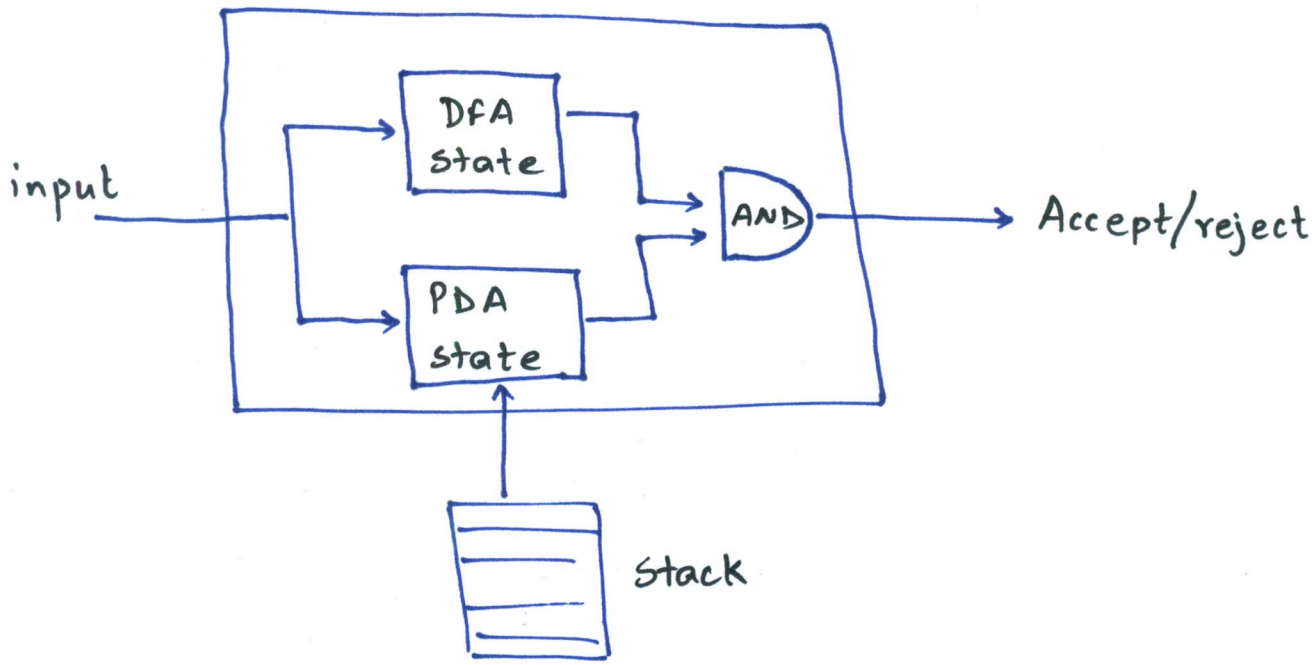
Theorem:

If L is a CFL and R is a regular set, then $L \cap R$ is a CFL

Proof: Let $L = L(M)$ for PDA $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$ that accepts M by final state

Let $R = L(A)$ for DFA $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$

We construct a PDA M' for $L \cap R$ by "running M and A in parallel" as follows:



formally

$$M' = (Q_A \times Q_M, \Sigma, \Gamma, \delta, [p_0, q_0], z_0, F_A \times F_M)$$

where $\delta([p,q], a, X)$ is defined to be the set of all pairs $([p', q'])$ such that

i. $p' = \delta_A(p, a)$

ii. pair $(q', \gamma) \in \delta_M(q, a, X)$

i.e.

- M' simulates moves of M on input ε without changing the state of A
- When M' simulates a move on input symbol a, M simulates that move and also simulates A's change of state on input a
- M' accepts iff both A and M accept

An induction on i shows that

$$([p_0, q_0], w, z_0) \vdash_{M'}^{i} ([p,q], \varepsilon, \gamma)$$

iff

$$(q_0, w, z_0) \vdash_{M}^{i} (q, \varepsilon, \gamma) \text{ and } \hat{\delta}_A(p_0, w) = p$$

Base: $i = 0$ is trivial since $p = p_0$, $q = q_0$, $\gamma = z_0$, $w = \varepsilon$

Induction:

Assume the statement for $i-1$ and let

$$([p_0, q_0], xa, z_0) \vdash_{M}^{i-1} ([p', q'], a, \beta) \vdash_{M'} ([p,q], \varepsilon, \gamma)$$

when $w = xa$ and a is ε or a symbol of Σ

By induction hypothesis

$$\hat{\delta}_A(p_0, x) = p' \text{ and } (q_0, x, z_0) \vdash_{M}^{i-1} (q, \varepsilon, \beta)$$

By definition of δ, the fact that

$([p', q'], a, \beta) \vdash_{M'} ([p, q], \varepsilon, r)$ tells us that

$\delta_A(p', a) = p$ and $(q', a, \beta) \vdash_M (q, \varepsilon, r)$

Thus, $\delta_A(p_0, w = xa) = p$ and $(q_0, w = xa, z_0) \vdash_M^i (q, \varepsilon, r)$

The converse can be shown similarly.

$\hookrightarrow (q_0, w, z_0) \vdash_M^i (q, \varepsilon, r)$ and $\delta_A(p, a, w) = \phi$

imply

$([p_0, q_0], w, z_0) \vdash_M^i ([p, q], \varepsilon, r)$

Theorem: $L_1 - L_2$ is not necessarily CFL when both L_1 and L_2 are CFLs.

Proof: Given CFLs L_1, L_2

- If $L_1 - L_2$ is a CFL, then $\Sigma^* - L$ would always be a CFL when L is a CFL, as we know Σ^* is CFL for every alphabet Σ
- But in that case $\overline{L} = \Sigma^* - L$ is a CFL when we pick proper alphabet Σ

which is a contradiction as CFLs are not closed under complementation.

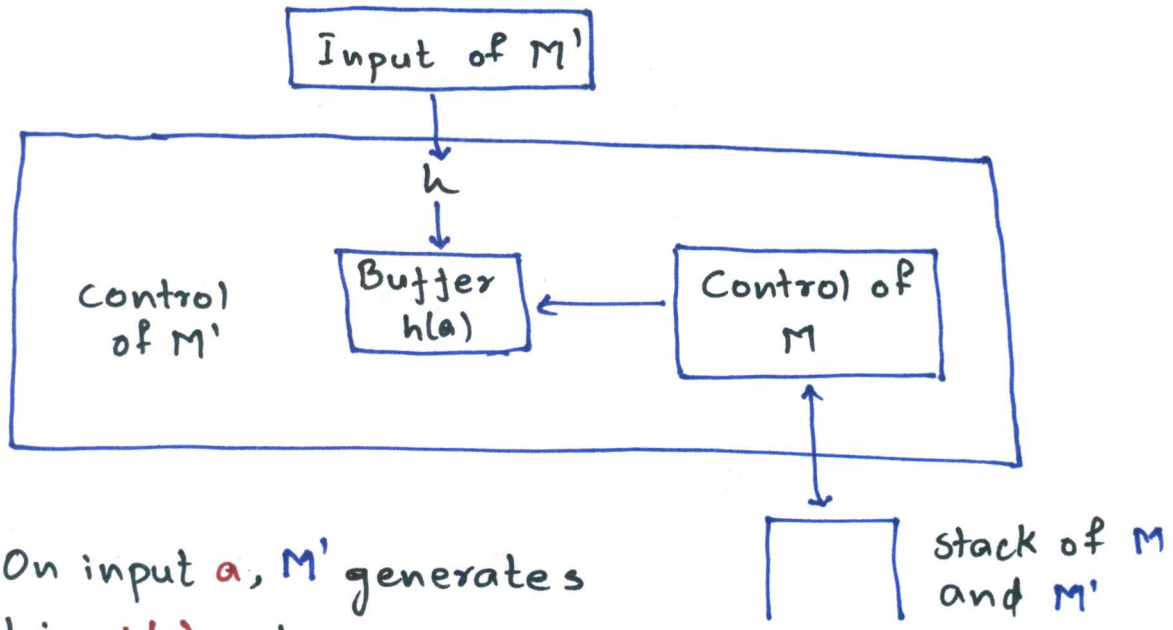
Hence $L_1 - L_2$ is not necessarily a CFL

Theorem:

The CFLs are closed under inverse homomorphism

Proof:

- $h: \Sigma \to \Delta$ homomorphism
- L CFL
- $L = L(M)$ where M is the PDA $(Q, \Delta, \Gamma, \delta, q_0, z_0, F)$
- Construct a PDA M' accepting $h^{-1}(L)$



- On input a, M' generates string $h(a)$ and simulates M on $h(a)$
- M' cannot necessarily simulate M's moves on $h(a)$ with one (or any finite number of) moves of its own as in the PDA case
  - (i) M could pop many symbols on a string
  - (ii) since it is non-deterministic, make moves that push an arbitrary number of symbols on the stack.
- M' uses a buffer, in which it may store $h(a)$
- M' is permitted to read a new input symbol only when the buffer is empty

- Buffer holds a suffix of $h(a)$ for some a at all times
- M' accepts its input w if buffer is empty and M is in a final state, i.e. M has accepted $h(w)$

$\therefore L(M') = \{w \mid h(w) \in L\}$

$= h^{-1}(L(M))$

Formal Construction

$M = (Q, \Delta, \Gamma, \delta, q_0, z_0, F)$ PDA for $L = L(M)$

↓ construct

$M' = (Q', \Sigma, \Gamma, \delta', [q_0, \varepsilon], z_0, F \times \{\varepsilon\})$ where

- Q' consists of pairs $[q, x]$ such that $q \in Q$ and x is a (not necessarily proper) suffix of $h(a)$ for some $a \in \Sigma$
- δ' is defined as follows

(i) $\delta'([q,x], \varepsilon, Y)$ contains all $([p,x], r)$ such that

$\delta(q, \varepsilon, Y)$ contains (p, r) | simulates ε-moves of M independent of the buffer contents

(ii) $\delta'([q,ax], \varepsilon, Y)$ contains all $([p,x], r)$ such that

$\delta(q, a, Y)$ contains (p, r) | simulates moves of M on input $a \in \Delta$, removing a from the front of the buffer

(iii) $\delta'([q,\varepsilon], a, Y)$ contains $([q, h(a)], Y)$ for all $a \in \Sigma, Y \in \Gamma$

- load buffer with $h(a)$, reading a from M's input
- the state and stack of M remains unchanged

Claim: $L(M') = h^{-1}(L(M))$

- if $(q, h(a), \alpha) \vdash_M^* (p, \varepsilon, \beta)$

  then $([q, \varepsilon], a, \alpha) \vdash_{M'}^* ([q, h(a)], \varepsilon, \alpha)$

  $\vdash_{M'}^* ([p, \varepsilon], \varepsilon, \beta)$

Thus if M accepts $h(w)$, that is

$(q_0, h(w), z_0) \vdash_M^* (p, \varepsilon, \beta)$

for some $p \in F$ and $\beta \in \Gamma^*$, it follows that

$([q_0, \varepsilon], w, z_0) \vdash_{M'}^* ([p, \varepsilon], \varepsilon, \beta)$

So M' accepts w

$\Rightarrow h^{-1}(L(M)) \subseteq L(M')$ —— (1)

Conversely, suppose M' accepts $w = a_1 a_2 \dots a_n$

Note: rule (iii) can be applied only with buffer empty

$\therefore ([q_0, \varepsilon], a_1 a_2 \dots a_n, z_0) \vdash_{M'}^* ([p_1, \varepsilon], a_1 a_2 \dots a_n, \alpha)$

rule 1: Simulate ε-moves of M independent of buffer contents.

$\vdash_{M'}^* ([p_1, h(a_1)], a_2 \dots a_n, \alpha_1)$

rule 3: load buffer with $h(a_1)$, a_1 from M's input

p_1, α_1 unchanged

$\vdash_{M'}^* ([p_2, \varepsilon], a_2 a_3 \dots a_n, \alpha_2)$

$\vdash_{M'}^* ([p_2, h(a_2)], a_3 \dots a_n, \alpha_2)$

$\vdash_{M'}^* ([p_2, h(a_3)], a_4 \dots a_n, \alpha_3)$

$$\vdots$$

$$\vdash^{*}_{M'} \left([p_{n-1}, \varepsilon], a_n, \alpha_n \right)$$

$$\vdash^{*}_{M'} \left([p_{n-1}, h(a_n)], \varepsilon, \alpha_n \right)$$

$$\vdash^{*}_{M'} \left([p_n, \varepsilon], \varepsilon, \alpha_{n+1} \right)$$

where $p_n \in F$

Thus $(q_0, \varepsilon, z_0) \vdash^{*}_{M} (p_1, \varepsilon, \alpha_1)$

and $(p_i, h(a_i), \alpha_i) \vdash^{*}_{M} (p_{i+1}, \varepsilon, \alpha_{i+1}) \quad \forall i$

Putting these moves together, we have

$$(q_0, h(a_1 a_2 \ldots a_n), z_0) \vdash^{*}_{M} (p_n, \varepsilon, \alpha_{n+1}), \; p_n \in F$$

$$\Rightarrow h(a_1 a_2 \ldots a_n) \in L(M)$$

$$\Rightarrow L(M') \subseteq h^{-1}(L(M)) \quad \text{—— (2)}$$

Combining (1) and (2), we have

$$L(M') = h^{-1}(L(M))$$

Use of closure properties (to prove certain languages are not CFL)

Example:

$L = \{w \in \{a,b,c\}^* \mid w \text{ has an equal no. of } a\text{'s}, b\text{'s}, c\text{'s}\}$ is not CFL

Let $L_1 = a^*b^*c^*$, regular set

$\therefore$ if L is CFL, then $L \cap L_1$ will also be CFL.

Now $L \cap L_1 = \{a^n b^n c^n : n \geq 0\}$ is already shown to be non-CFL.

Hence L is not regular

Example: Let $L = \{ww \mid w \text{ is in } (a+b)^*\}$ is not CFL

i.e. L consists of all words whose first and last halves are the same, is not CFL.

Soln:

Let L be CFL

$a^+b^+c^+d^+ \rightarrow$ regular language

$\therefore L = L \cap a^+b^+c^+d^+ = \{a^i b^j a^i b^j \mid i \geq 1, j \geq 1\}$ is CFL.

Let h be a homomorphism defined by

$h(a) = h(c) = a$, $h(b) = h(d) = b$

Then, $h^{-1}(L_1) = \{x_1x_2x_3x_4 \mid x_1x_3 \in (a+c)^+$ are of same length and $x_2, x_4 \in (b+d)^+$ are of same length$\}$

$\therefore a^*b^*c^*d^* \cap h^{-1}(L_1) = \underline{L_2} = \{a^i b^j c^i d^j \mid i \geq 1, j \geq 1\}$ must be CFL.

But we have already shown that L_2 is not CFL,

Therefore L cannot be CFL

Decision Algorithms for CFLs

Theorem: There are algorithms to determine if a CFL is (a) empty (b) finite or (c) infinite

Proof:

(a) $L(a)$ non-empty iff the start symbol S generates some strings of terminals

- can be efficiently tested
- Algm 1 determine set of variables W when each $A \in W$ derives a terminal string
  → if $S \in W$, then CFL is non-empty

(b) or (c)

$L(a)$ is finite iff $L(G')$ is finite when G' is the CNF grammar with no useless symbols, generating $L(a) - \{\varepsilon\}$

Draw a directed graph with

- each variable is a vertex
- AB is an edge if $A \to BC$ or $A \to CB$ is production for any C.

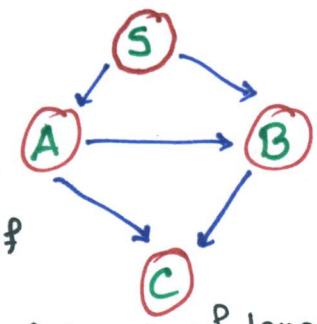
Example:

$S \to AB,\ A \to BC \mid a,\ B \to CC \mid b,\ C \to a$

Claim: $L(G')$ is finite iff this graph is acyclic

Rank of a variable A → the length of the longest path in this graph beginning at A.

Vertex	Rank
$S = v_0$	3
A	2
B	1
C	0



No cycle

⇒ Rank of every variable is finite

- Rank of $S = v_0$ → not more than # of variables
  → S derives no. of strings of length $> 2^{v_0}$
  → language is finite

Claim: $L(G')$ is finite iff this graph is acyclic

Proof:

- If there is a cycle, say $A_0 A_1 \dots A_n A_0$ then

$$A_0 \Rightarrow \alpha_1 A_1 \beta_1 \Rightarrow \alpha_2 A_2 \beta_2 \Rightarrow \dots \Rightarrow \alpha_n A_n \beta_n \Rightarrow \alpha_{n+1} A_0 \beta_{n+1}$$

where the α's and β's are strings of variables with $|\alpha_i \beta_i| = i$ (each time, only one variable/symbol increase)

Since there are no useless symbols,

$$\alpha_{n+1} \overset{*}{\Rightarrow} w \text{ and } \beta_{n+1} \overset{*}{\Rightarrow} x \text{ for some } w, x \in T^*$$

of total length at least $n+1$

Since $n \geq 0$, w and x cannot both be ε

Also, as there are no useless symbols, we can find $y, z \in T^*$ such that $S \overset{*}{\Rightarrow} y A_0 z$ (A_0 in some sequential form)

and also we can find $v \in T^*$ such that

$$A_0 \overset{*}{\Rightarrow} v \quad (A_0 \text{ derives a terminal string})$$

Then $\forall i$

$$S \overset{*}{\Rightarrow} y A_0 z \overset{*}{\Rightarrow} y w A_0 x z \overset{*}{\Rightarrow} y w^2 A_0 x^2 z \overset{*}{\Rightarrow} \dots$$
$$\dots \overset{*}{\Rightarrow} y w^i v x^i z$$

As $|wx| > 0$, $y w^i v x^i z \neq y w^j v x^j z$ if $i \neq j$

$\Rightarrow$ The grammar generates an infinite no. of strings

- Conversely, suppose the graph has no cycle

Define the rank of a variable A to be

- the length of the longest path in the graph begining at A

- the absence of cycle implies the rank of A is finite

Now if $A \to BC$ is a production, then rank of B and rank of $C <$ the rank of A,

because for every path from B or C ∃ a path of length one greater from A

Claim: if rank of $A = r$, then no terminal string derived from A has length $> 2^r$

Proof (By induction)

Basis: $r = 0$

If rank of A is 0, then it's vertex has no edge cut

∴ All A-productions have terminals on the right, and A derives only string of length 1.

Induction $r > 0$

If we use a production of the form $A \to a$, we may derive only a string of length 1.

If we begin with $A \to BC$, then as B, C are of rank $r-1$ or less, by induction hypothesis, they derive only strings of length 2^{r-1} or less.

Thus BC cannot derive a string of length $> 2^r$

Since, S is of finite rank r_0, and infact, is of rank no greater than the number of variables (as no cycle), S derives strings of length no greater than 2^{r_0}

Thus, the language is finite.

Membership

Input: A CFG $G = (V, T, P, S)$ a string $x \in T^*$

Output: YES if $x \in L(G)$, NO otherwise

$$G \xrightarrow{\text{GNF}} G' = (V', T, P', S)$$

such that $L(G') = L(G) - \{\varepsilon\}$

case: $x = \varepsilon$ → already discussed

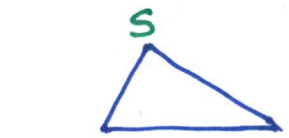
case: $x \neq \varepsilon$

- every production in GNF grammar adds exactly one terminal to the string being generated
- So if x has a derivation in G' it has one with exactly $|x|$ steps
- If no variable of G' has more than productions, then there are atmost $K^{|x|}$ leftmost derivations of strings of length $|x|$
- Inefficient algorithm, exponential time algorithm in $|x|$

Cocke-Younger-Kasami (CYK) Algorithm

Input: $x = a_1 a_2 \cdots a_n \in T^*$, CFG $G = (V, T, P, S)$

Output: Check if $x \in G$ i.e. $S \overset{*}{\Rightarrow} x = a_1 \cdots a_n$



$x = a_1 \cdots \underbrace{a_i \cdots a_{i+j-1}} \cdots a_n$

substring of x of length j begining at position i.

$x_{ij} = a_i a_{i+1} \cdots a_{i+j-1}$

- $V_{ij} = \{ A \in V \mid A \overset{*}{\Rightarrow} x_{ij} \}$

  $= \{ A \in V \mid A \to BC$ is in P with $B \in V_{ik}$ and $C \in V_{i+k, j-k}$ for $1 \le k < j \}$

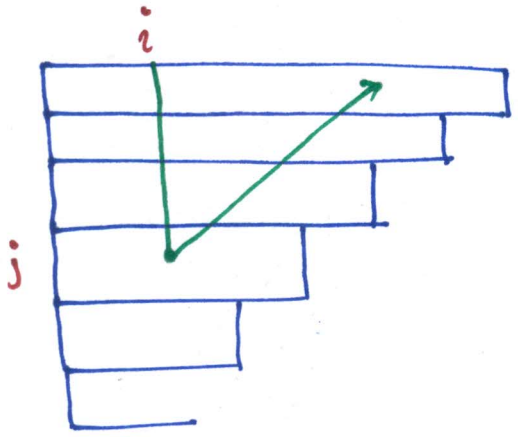
- $V_{1n} = \{ A \in V \mid A \overset{*}{\Rightarrow} x_{1n} = a_1 \cdots a_n = x \}$

  Check if $S \in V_{1n}$

- $V_{i1} = \{ A \in V \mid A \overset{*}{\Rightarrow} x_{i1} = a_i$ i.e. $A \to a_i$ is in $P \}$

- Use dynamic programming (table filling method) to compute all V_{ij}, $j = 2, \ldots, n$; $i = 1, 2, \ldots, n-j+1$

  $K = 1, \ldots, j-1$



```
for i = 1 to n do
    V_i1 = { A ∈ V | A → a_i is in P }
for j = 2 to n do
    for i = 1 to n-j+1 do
        V_ij = ϕ
        for k = 1 to j-1 do
            V_ij = V_ij ∪ { A ∈ V | A → BC is in P, B is in
                                V_ik and C is in V_i+k,j-k }
        end do
    end do
end do.
```

Example: Consider the CFG G

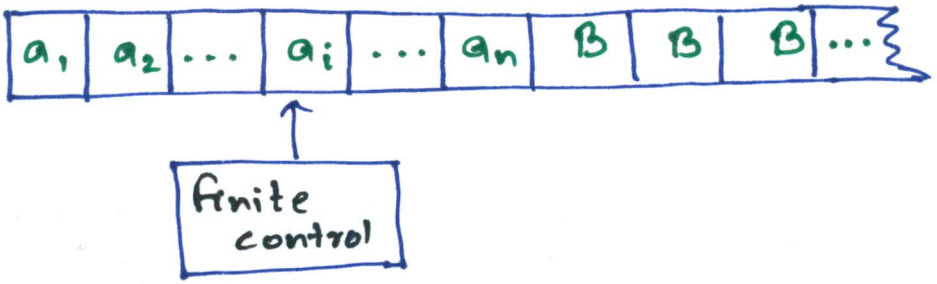
$S \to AB \mid BC$, $A \to BA \mid a$, $B \to CC \mid b$, $C \to AB \mid a$

and the input baaba

B	A,C	A,C	B	A,C
A,S	B	S,C	A,S	
ϕ	B	B		
ϕ	S,A,C			
S,A,C				

Turing Machine

- A finite control, an input tape that is divided into cells, and a tape head than scans one cell of the tape at a time
- The tape has leftmost cell, but is infinite to the right
- Each cell of the tape may store exactly one symbol
- R/W tape head can examine one cell at a time
- n leftmost cells initially hold the input for some finite $n \geq 0$, remaining infinity of cells each hold the blank, which is a special tape symbol that is not an input symbol.



- In one move the Turing machine, depending upon the symbol scanned by the R/W tape head and the state of the finite control
  1. changes state
  2. prints a symbol on the tape cell scanned, replacing what was written there, and
  3. moves its head left or right one cell
- Unlike 2-way FA, Turing machine has the ability to change symbols on its tape.

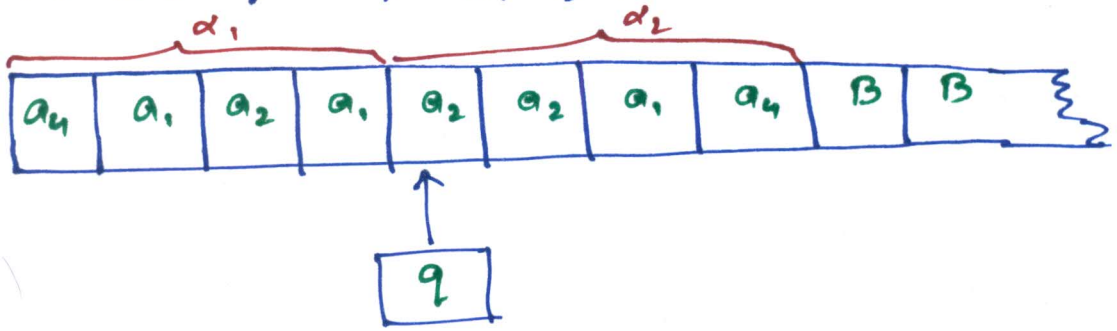
Formally a Turing Machine M is a 7-tuple

$(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

1. Q is a finite non-empty set of states
2. Γ is a finite non-empty set of tape symbols
3. $B \in \Gamma$ is the blank
4. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
   (δ may be undefined for some arguments)
5. $q_0 \in Q$ is the initial state
6. $F \subseteq Q$ is the set of final states

Instantaneous Description (ID)

- $\alpha_1 q \alpha_2$, $q \in Q$, $\alpha_1, \alpha_2 \in \Gamma^*$



- B may occur in α_1, α_2

Moves in a Turing Machine

- $\delta(q, x)$ induces a change in ID of TM
- $X_1 X_2 \ldots X_{i-1}\, q\, X_i \ldots X_n$ be an ID and let

$$\delta(q, X_i) = (p, Y, L)$$

where if $i-1 = n$ then $X_i = B$ and if $i = 1$, then there is no next ID

Move relation $\vdash_M$

if $i > 1$

$X_1 X_2 \cdots X_{i-1}\, q\, X_i \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-2}\, p\, X_{i-1} Y \cdots X_r$

suffix deleted if completely blank.

if $i = 1$

resulting ID is $pY X_{i+1} \cdots X_n$

- $\delta(q, X_i) = (p, Y, R)$

$X_1 X_2 \cdots X_{i-1}\, q\, X_i \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-1}\, Y\, p\, X_{i+1} \cdots X_n$

if $i-1 = n$, the string $X_i \cdots X_n$ is empty and the resulting ID is $X_1 X_2 \cdots X_{i-1}\, p$

if $i = n$, then resulting ID is $X_1 X_2 \cdots X_{n-1}\, Y\, p\, B$

- $I_j \vdash I_k$ defines a relation among IDs
- $\vdash^*$ reflexive transition closure of $\vdash$
- Languages accepted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$$L(M) = \{ w \in \Sigma^* \mid q_0 w \vdash^* \alpha_1 p \alpha_2 \text{ for some } p \in F, \text{ and } \alpha_1, \alpha_2 \in \Gamma^* \}$$

- Given a TM recognizing a language L, we assume without loss of generality that the TM halts, i.e. has no next move whenever the input is accepted.

for words not accepted by TM, it is possible the TM will never halt (or halts in non-accepting state)

Example:

Consider the TM M described by the following transition table. Describe the processing of (a) 011 (b) 0011 (c) 001 using IDs. Which of the above strings are accepted by M

present state	Tape symbol				
	0	1	x	y	B
$\rightarrow q_1$	xRq_2				BRq_6
q_2	$0Rq_2$	yLq_3		yRq_2	
q_3	$0Lq_4$		xRq_5	yLq_3	
q_4	$0Lq_4$		xRq_1		
q_5				yRq_5	BRq_6
(q_6)					

a). $q_1011 \vdash xq_211 \vdash q_3xy1 \vdash xq_5y1 \vdash xyq_51$

$\delta(q_5, 1)$ not defined, M halts.

b) $q_10011 \vdash xq_2011 \vdash x0q_211 \vdash xq_30y1 \vdash q_4x0y1$
$\vdash xq_10y1 \vdash xxq_2y1 \vdash xxyq_21 \vdash xxq_3yy$
$\vdash xq_3xyy \vdash xxq_5yy \vdash xxyq_5y \vdash xxyyq_5B$
$\vdash xxyyBq_6$

M halts as q_6 is an accepting state.

c) $q_1001 \vdash xq_201 \vdash x0q_21 \vdash xq_30y \vdash q_4x0y \vdash xq_10y$
$\vdash xxq_2y \vdash xxyq_2$

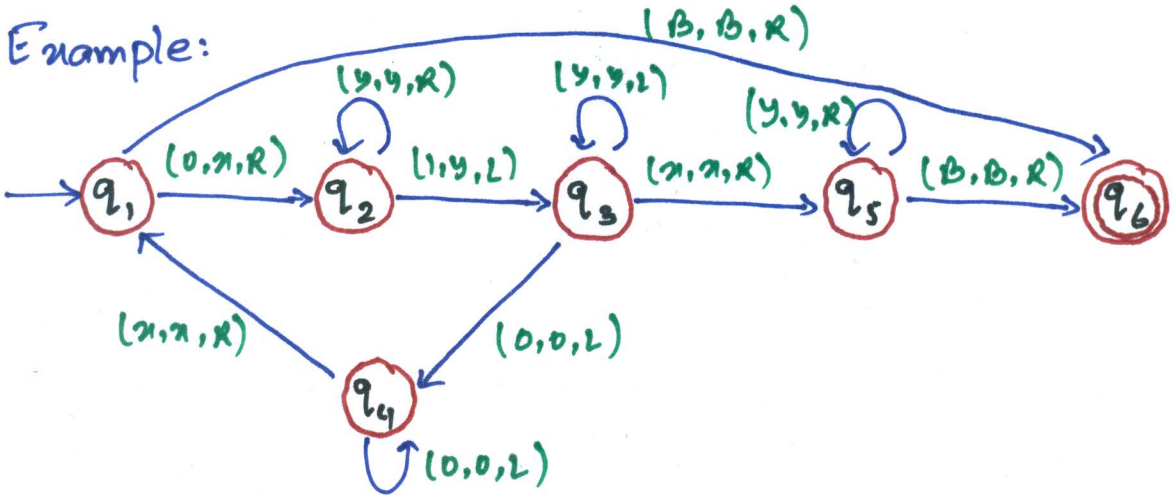
M halts, as q_2 is not accepting state $001 \notin L(M)$

Transition Diagram

- vertices represents state
- directed edges represents transition of states

  when $\delta(q_i, a) = (q_j, b, L)$, there is a directed edge from q_i to q_j with label (a, b, L)

Example:



Obtain the computation sequence of M for processing the input string 0011

Soln:

$q_1 0011 \vdash x q_2 011 \vdash x0 q_2 11 \vdash x q_3 0y1 \vdash q_4 x0y1$

$\vdash x q_1 0y1 \vdash xx q_2 y1 \vdash xxy q_2 1 \vdash xx q_3 yy$

$\vdash x q_3 xyy \vdash xx q_5 yy \vdash xxy q_5 y \vdash xxyy q_5 B$

$\vdash xxyyB q_6$

Example: Design a TM that accepts $\{0^n 1^n \mid n \geq 1\}$

Soln:

input string w

i) if the leftmost symbol of w is 0, replace it by x and move right till a leftmost 1 is encountered in w Change it to y and move backward.

ii) Repeat (i) with the leftmost 0

If move back and forth and no 0 or 1 remains, move to a final state.

iii) for strings not in the form $0^n 1^n$, the resulting state has to be non-final.

Construction of TM M

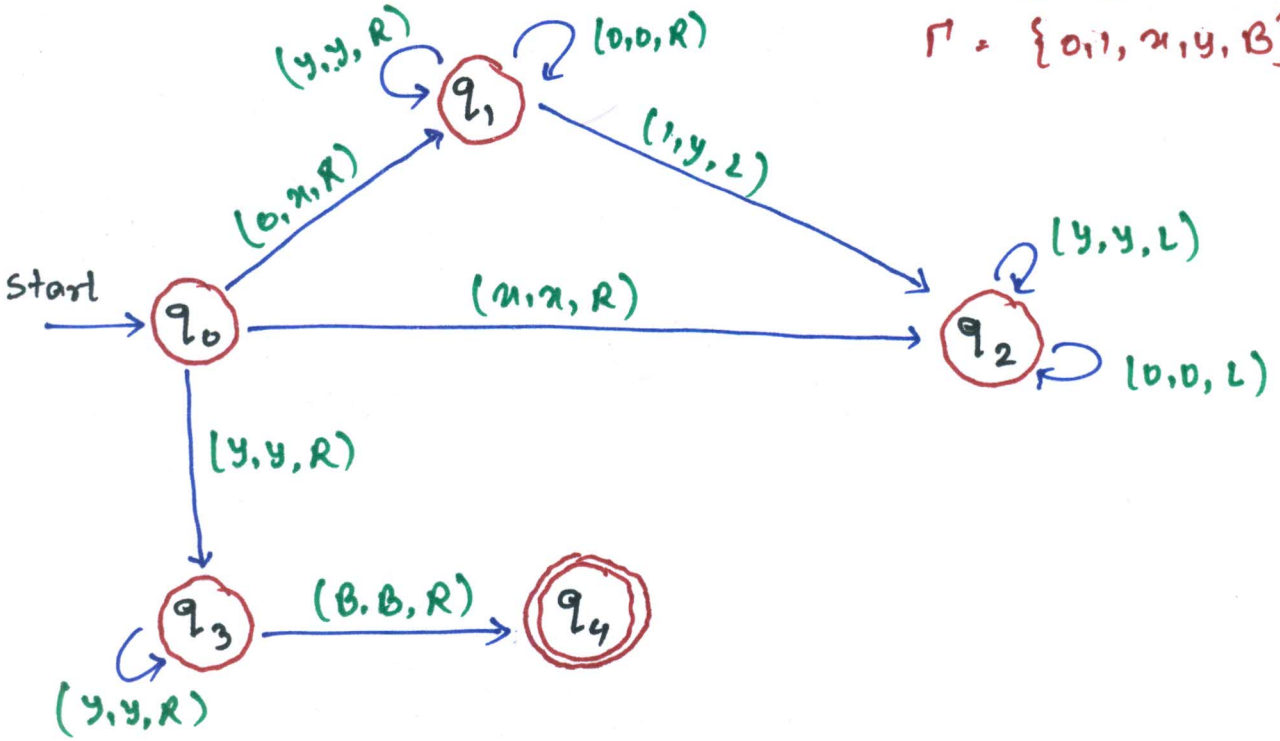
$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$F = \{q_4\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, x, y, B\}$

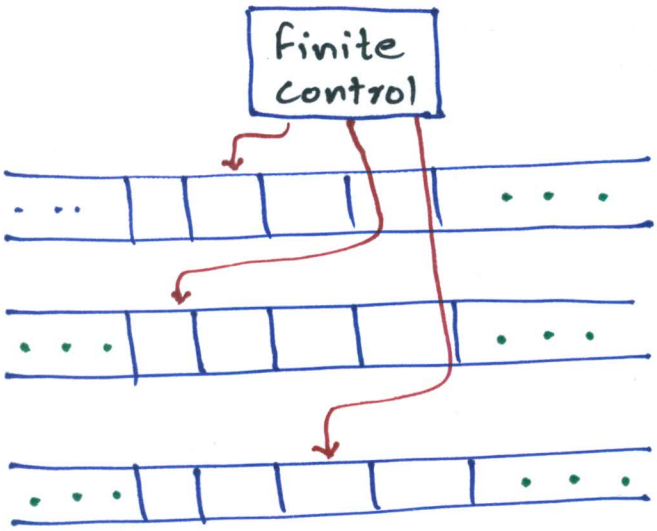


010: $q_0 010 \vdash x q_1 10 \vdash q_2 x y 0 \vdash x q_0 y 0 \vdash x y q_3 0$

M halts as $\delta(q_3, 0)$ is undefined and 010 is not accepted by M

Variants of Turing Machine

- **Multitape TM**



$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

- Non-deterministic TM

$\delta(q,a) = \{(p_1, y_1, D_1), (p_2, y_2, D_2), \ldots, (p_n, y_n, D_n)\}$

$p_1, p_2, \ldots, p_n \in Q$

$y_1, y_2 \ldots y_n \in \Gamma$

$D_i = R$ or L

$\delta: Q \times \Gamma \rightarrow$ a subset of $Q \times \Gamma \times \{L, R\}$

- Every language accepted by a ~~multiple~~ multitape/ nondeterministic TM is accepted by some single tape TM (i.e. standard TM)

Running Time of a TM (when the TM halts)

- # of steps the TM takes before halting on input string w
- $T(n)$ = maximum of running time of the TM over all inputs of w size n

- $M_1 \rightarrow$ single tape TM simulating multitape TM M

  M_1 takes $O(n^2)$ time to simulate n moves of M

- $L \subseteq \Sigma^*$ recursively enumerable if $\exists$ a TM M such that

$$L = T(M)$$

- $L \subseteq \Sigma^*$ recursive if $\exists$ some TM M satisfying
  - i. if $w \in L$ then M accepts w and halts
  - ii. if $w \notin L$ then M eventually halts, without reaching to an accepting state

Decidable problems (Yes/No) → if the corresponding language is recursive

- $A_{DFA} = \{ (B, w) \mid \text{DFA } B \text{ accepts } w \}$ → decidable
- $A_{CFG} = \{ (G, w) \mid \text{CFG } G \text{ accepts } w \}$ → decidable
- $A_{CSG} = \{ (G, w) \mid \text{Context sensitive Grammar } G \text{ accepts } w \}$ → decidable

- CSG: Context sensitive Grammar
  (consists of all type 1 productions)
  - type 0 grammar: no restrictions on the productions
  - type 1 grammar: $\phi A \psi \rightarrow \phi \alpha \psi$, $\alpha \neq \varepsilon$
    ϕ is the left context
    ψ is the right context
  - type 2 grammar: $A \rightarrow \alpha$, $A \in V$, $\alpha \in (V \cup T)^*$
  - type 3 grammar: $A \rightarrow a$ or $A \rightarrow aB$, $a \in T$, $A, B \in V$
    ($S \rightarrow \varepsilon$ is allowed, provided S doesnot appear on R.H.S. of any production)

Undecidable Problems

- $A_{TM} = \{ (M, w) \mid \text{The TM } M \text{ accepts } w \}$
- $HALT_{TM} = \{ (M, w) \mid \text{The TM } M \text{ halts on } w \}$

Church Turing Thesis

Given any problem that can be solved with an effective algorithm, there is a TM that can solve this problem

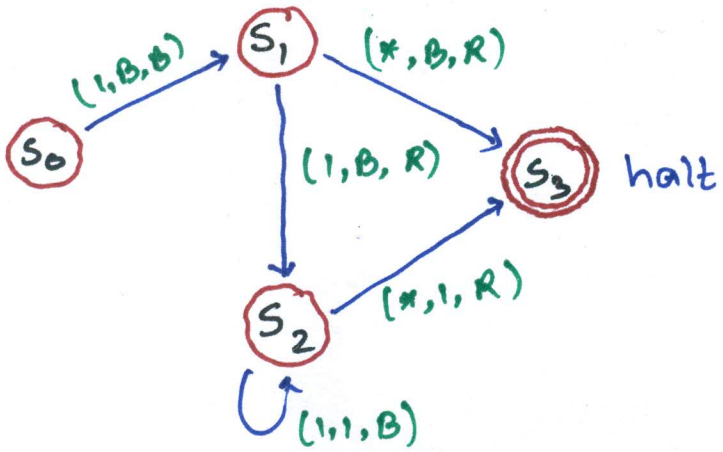
- Tremendous amount of evidence exists for Church - Turing Thesis
- it may be difficult to construct a TM to compute a particular function that can be computed by an algorithm
  → such a TM can always be found
- Algorithm ↔ TM

- integer n → represented as → $n+1$ 1's
  0 → one 1
  5 → 111 111 (6 one's)
- 4 touple $(2,0,1,3)$ = 111 * 1 * 11 * 1111

Example: TM for $f(n_1, n_2) = n_1 + n_2$

⇒ n_1+1 1's * n_2+1 1's

3+2 = 5



Unsolvable Problems (no effective algorithm exists to solve a problem)

- Halting Problem
- Problem of determining whether two CFGs generate same set of strings
- Hilbert's 10^{th} problem
  - whether there are integers solutions to a given polynomial equation with integer co-efficient
  - 23 problems Hilbert posed in 1990
    - Progress of 20^{th} century mathematics

Post's Correspondence Problem (PCP)

- Σ an alphabet having atleast two elements
- Given two ordered sets of strings $w_1, w_2, \ldots, w_K$ and $x_1, x_2, \ldots, x_K$ over Σ,

is it possible to find integers. $i_1, i_2, \ldots, i_n$, $1 \leq i_j \leq K$ such that

$$w_{i_1} w_{i_2} \ldots w_{i_n} = x_{i_1} x_{i_2} \ldots x_{i_n} \; ?$$

- Halting problem undecidable

  using this → PCP is undecidable

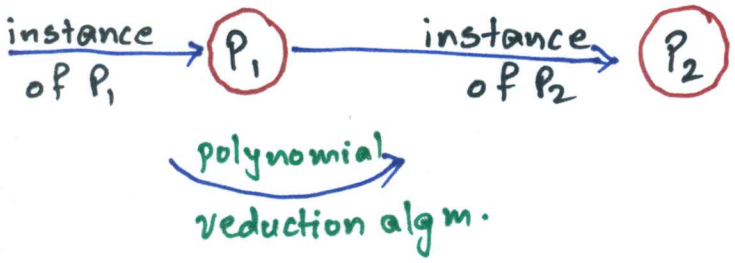
  using this → Is CFG ambiguous or undecidable?

The classes P and NP

- $T(n)$ is the time complexity of a TM M
  - given an input w of length n, M halts after making at most $T(n)$ moves

- L is in $P \rightarrow L = L(M)$ for some polynomial time deterministic TM M
  (i.e. $T(n)$ is polynomial in n)
- L is in $NP \rightarrow L = L(M)$ for some polynomial time non-deterministic TM M
  (i.e. M executes at most $T(n)$ moves for every input w of length n, $T(n)$ being a polynomial in n)

Polynomial reduction



- If we have a polynomial reduction from P_1 to P_2 and P_2 is in P, then P_1 must be in P

Non-complete Problem

- L is NP-complete if
  (i) L is in NP
  (ii) for every L' in NP $\exists$ a polynomial time reduction of L' to L

- If a problem is NP-complete, then there is no need to find polynomial or easy algorithm for the problem
  - saves time and energy

- SAT is NP-complete

  SAT = { Boolean function f | $\exists$ some truth assignment t s.t. $f(t) = 1$ }

- Knapsack problem
  - Given a set $A = \{a_1, \ldots a_n\}$ of non-negative integers and an integer K does there exist $B \subset A$ such that

$$\sum_{b_j \in B} b_j = K \; ?$$

- CSAT
  - Given a Boolean expression in CNF, is it satisfiable?
- Hamiltonian Circuit Problem
  - Does a graph G have a Hamiltonian Circuit (circuit passing through each edge of G exactly once)
- Travelling Salesman Problem (TSP)
  - Given n cities, the distance between them and a number D, does there exist a tour program for a salesman to visit all the cities exactly once so that the distance travelled is at most D
- Vertex Cover Problem
  - Given a graph G and a natural number K, does there exist a vertex cover for G of K vertices.

  (Vertex cover of G is a subset C of vertices of G such that each edge of G has an odd vertex in C)