Chapter 3

Pushdown Automata and Context-Free Languages

NPDAs

- A NPDA (Nondeterministic PushDown Automata) is a 7-tuple

$M = (Q,\Sigma,\Gamma, \delta ,s, \perp , F)$ where

- Q is a finite set (the states)
- Σ is a finite set (the input alphabet)
- Γ is a finite set (the stack alphabet)
- $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$ is the transition relation
- $s \in Q$ is the start state
- $\perp \in \Gamma$ is the initial stack symbol
- $F \subseteq Q$ is the final or accept states

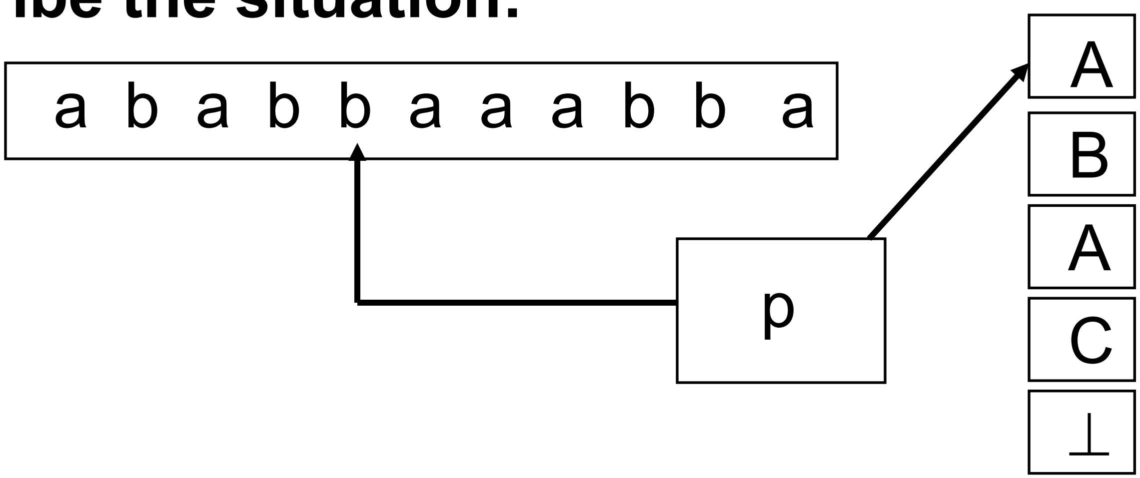
- $((p,a,A),(q,B_1B_2\ldots B_k)) \in \delta$ means that

whenever the machine is in state p reading input symbol a on the input tape and A on the top of the stack, it pops A off the stack, push $B_1B_2\ldots B_k$ onto the stack (B_k first and B_1 last), move its read head right one cell past the one storing a and enter state q.

$((p,\varepsilon,A),(q,B_1B_2\ldots B_k)) \in \delta$ means similar to $((p,a,A),(q,B_1B_2\ldots B_k)) \in \delta$ except that it need not scan and consume any input symbol.

Configurations

- **Collection of information used to record the snapshot of an executing NPDA**
- **an element of $Q \times \Sigma^* \times \Gamma^*$.**
- **Configuration $C = (q, x, w)$ means**
  - **the machine is at state q,**
  - **the rest unread input string is x,**
  - **the stack content is w.**
- **Example: the configuration (p, baaabba, ABAC⊥) might describe the situation:**



Start configuration and the next configuration relations

- Given a NPDA M and an input string x, the configuration $(s, x, \perp)$ is called the start configuration of NPDA on x.
- $CF_M =_{def} Q \times \Sigma^* \times \Gamma^*$ is the set of all possible configurations for a NPDA M.
- One-step computation of a NPDA:
  - Let the next configuration relation -->_M on CF_M be the set of pairs :
  - $\{ (p, ay, A\beta) \text{-->}_M (q, y, \gamma\beta) \mid ((p,a,A), (q, \gamma)) \in \delta. \} \cup$
  - $\{ (p, y, A\beta) \text{-->}_M (q, y, \gamma\beta) \mid ((p,\varepsilon,A), (q, \gamma)) \in \delta \}$
  - -->_M describes how the machine can move from one configuration to another in one step. (i.e., C -->_M D iff D can be reached from C by executing one instruction)
  - Note: NPDA is nondeterministic in the sense that for each C there may exist multiple D's s.t. C -->_M D.

Multi-step computations and acceptance

- **Given a next configuration relation -->_M:**

  Define --->^n_M and --->^*_M as usual, i.e.,

  - $C \text{-->}^0_M D$ iff $C = D$.
  - $C \text{-->}^{n+1}_M$ iff $\exists E$ $C \text{-->}^n_M E$ and $E \text{-->}_M D$.
  - $C \text{-->}^*_M D$ iff $\exists n \geq 0$ $C \text{-->}^n_M D$.
  - i.e., --->^*_M is the ref. and trans. closure of -->_M .
- **Acceptance: When will we say that an input string x is accepted by an NPDA M?**
  - two possible answers:
  - 1. by final states: M accepts x (by final state) iff
  - $(s,x,\perp) \text{-->}^*_M (p,\varepsilon,\alpha)$ for some final state $p \in F$.
  - 2. by empty stack: M accepts x by empty stack iff
  - $(s,x,\perp) \text{-->}^*_M (p,\varepsilon,\varepsilon)$ for any state p.
  - Remark: both kinds of acceptance have the same expressive power.

Language accepted by a NPDAs

$M = (Q,\Sigma,\Gamma,\delta,s,,F)$: a NPDA.

The languages accepted by M is defined as follows:

- **1. accepted by final state:**
- **$L_f(M) = \{x \mid M$ accepts x by final state$\}$**
- **2. accepted by empty stack:**
- **$L_e(M) = \{x \mid M$ accepts x by empty stack$\}$.**
- **3. Note: Depending on the context, we may sometimes use L_f and sometimes use L_e as the official definition of the language accepted by a NPDA. I.e., if there is no worry of confusion, we use $L(M)$ instead of $L_e(M)$ or $L_f(M)$ to denote the language accepted by M.**
- **4. In general $L_e(M) \neq L_f(M)$.**

Some example NPDAs

Ex 23.1 : Define a NPDA M_1 which accepts the set of balanced strings of parentheses [ ] by empty stack.

- M_1 requires only one state q and behaves as follows:
- repeat { 1. if input is '[' : push '[' onto the stack ;
- 2. if input is ']' and top is '[' : pop
- 3. if input is 'ε' and top is ⊥ : pop. }

Formal definition: $Q = \{q\}$, $\Sigma = \{[,]\}$, $\Gamma = \{[, \bot\}$,

start state = q, initial stack symbol = ⊥.

$\delta = \{\ ((q,[,\bot),(q,[\bot)),\ \ ((q,[,[),(q,[[)),$ // 1.1, 1.2

$((q,],[),\ (q,\varepsilon)),$ // 2

$((q,\varepsilon,\bot),(q,\varepsilon))\ \}$ // 3

Transition Diagram representation of the program δ :

$((p,a\ A),(q,B_1\ldots B_n)) \in \delta \Rightarrow$ $p \xrightarrow{a,A\,/B_1\ldots B_n} q$

- This machine is not deterministic. Why ?

Example : Execution sequences of M1

- Let input x = [[[ ]] [ ]] [ ]. Then below is a successful computation of M_1 on x:
- (q, [[[ ]] [ ]] [ ], ⊥) : the start configuration

transition (1) $\to_M$ (q, [[ ]] [ ]] [ ], [⊥)
transition (1) $\to_M$ (q, [ ]] [ ]] [ ], [[⊥)
(1) $\to_M$ (q,]] [ ]] [ ], [[[⊥)
(2) $\to_M$ (q,] [ ]] [ ], [[⊥)
(1) $\to_M$ (q, [ ]] [ ], [⊥)
(2) $\to_M$ (q,]] [ ], [[⊥)
(2) $\to_M$ (q,] [ ], [⊥)
(1) $\to_M$ (q, [ ], ⊥)
(2) $\to_M$ (q,], [⊥)
(2) $\to_M$ (q, , ⊥)
(3) $\to_M$ (q, ,) :accept configuration
accepts by empty stack

Failure computation of M1 on x

- **Note besides the above successful computation, there are other computations that fail.**

Ex: $(q, [[[]][]][], \quad \perp)$: the start configuration

$\to^*_M (q, [], \quad \perp)$

$\to_M (q, [], \quad)$ transition (3)

a dead state in which the input is not empty and we cannot move further ==> failure!!

Note: For a NPDA to accept a string x, we need *only one successful computation* (i.e., $\exists$ D = (_, ε, ε) with empty input and stack s.t. $(s,x,\perp) \to^*_M D$.)

- **Theorem 1: String $x \in \{[,]\}^*$ is balanced iff it is accepted by M_1 by empty stack.**

- **Definitions:**
  1. A string x is said to be pre-balanced if $L(y) \geq R(y)$ for all prefixes y of x.
  2. A configuration (q, z, α) is said to be blocked if the pda M cannot use up input z, i.e., there is no state r and stack β such that $(q, z, \alpha) \rightarrow^* (r, \varepsilon, \beta)$.
- **Facts:**
  - 1. If initial configuration $(s, z, \perp)$ is blocked then z is not accepted by M.
  - 2. If (q, z, α) is blocked then (q, zw, α) is blocked for all $w \in \Sigma^*$.

Pf: 1. If $(s, z, \perp)$ is blocked, then there is no state p, stack β such that $(s, z, \perp) \dashrightarrow^* (p, \varepsilon, \beta)$, and hence z Is not accepted.

2. Assume (q, zw, α) is not blocked, then there must exists intermediate cfg (p, w, α') such that $(q, zw, \alpha) \rightarrow^* (p, w, \alpha') \rightarrow^* (r, \varepsilon, \beta)$. But $(q, zw, \alpha) \rightarrow^* (p, w, \alpha')$ implies $(q, z, \alpha) \rightarrow^* (p, \varepsilon, \alpha'')$ and (q, z, α) is not blocked.

- **Lemma 1: For all strings z,x,**
  - if z is prebalanced then (q, zx,$\perp$)-->* (q,x, $\alpha\perp$) iff α = $[^{L(z)-R(z)}$;
  - if z is not prebalanced, (q, z, $\perp$) is blocked.

Pf: By induction on z.

basic case: z = ε. Then (q, zx,$\perp$) = (q, x,$\perp$) $\rightarrow^*$ (q,x, $\alpha\perp$) iff α = ε = $[^{L(z)-R(z)}$.

inductive case: z = ya, where a is '[' or ']'.

case 1: z = y[. If y is prebalanced, then so is z.

By ind. hyp., (q,y[x, $\perp$) -->* (q, [x, $[^{L(y)-R(y)}\perp$), hence

(q, zx, $\perp$) = (q,y[x, $\perp$) -->* (q, [x, $[^{L(y)-R(y)}\perp$)

-->(q, x, $[[^{L(y)-R(y)}\perp$) =(q, x, $[^{L(z)-R(z)}\perp$).

and if (q, zx, $\perp$) $\rightarrow$* (q,x, $\alpha\perp$), there must exists α' such that

(q, zx, $\perp$) = (q, y[x, $\perp$) $\rightarrow$* (q,[x, $\alpha'\perp$), $\rightarrow$* (q,x, $\alpha\perp$). But, by ind.hyp., α' = $[^{L(y)-R(y}$, hence the only allowable instruction is 1.1(push [), hence a = [a' = $[^{L(z)-R(z)}$.

If y is not prebalanced, then, by ind. hyp., (q, y, $\perp$) is blocked and hence (q, y[, $\perp$) is blocked as well.

case 2: z = y]. here are 3 cases to consider .

case 21: y is not prebalanced. Then z neither prebalanced.

By ind. hyp. $(q, y, \perp)$ is blocked, hence $(q, y], \perp)$ is blocked

case 22: y is prebalanced and L(y) = R(y). Then z is not prebalanced.

By ind. hyp., $(q, y],\perp) \rightarrow^* (q,], \alpha\perp)$ iff $\alpha = [^{L(z)-R(z)} = \varepsilon$.

But then $(q,],\perp)$ is blocked. Hence $(q, z,\perp)$ is blocked.

case23: y is prebalanced and L(y) > R(y). Then z is prebalanced as well.

By Ind.hyp., $(q, y],\perp) \rightarrow^* (q,], \alpha\perp)$ iff $\alpha = [^{L(z)-R(z)}$ matches $[^+$. Hence

$(q,y]x,\perp) \rightarrow^* (q,]x, [^{L(y)-R(y)} \perp)$ --- ind. hyp

$\rightarrow (q, x, [^{L(y)-R(y)-1}\perp)$ --- (instruction 2)

$= (q, x, [^{L(z)-R(z)}\perp)$

On the other hand, if $(q,y]x,\perp) \rightarrow^* (q,x, \alpha\perp)$.

Then there must exist a cfg $(q,]x, \alpha'\perp)$ such that

$(q,y]x,\perp) \rightarrow^* (q,]x, \alpha'\perp) \rightarrow^* (q,x, \alpha\perp)$., where, by ind.hyp., $\alpha' = [^{L(y)-R(y)}$.

I.e, $(q,y]x,\perp) \rightarrow^* (q,]x, [^{L(y)-R(y)}\perp) \rightarrow^* (q,x, \alpha\perp)$.

But then the only instruction executable in the last part is (2).

Hence $\alpha = [^{L(y)-R(y)-1} = [^{L(z)-R(z)}$.

Pf [of theorem 1] : Let x be any string.

If x is balanced, then it is prebalanced and $L(x) - R(x) = 0$.

Hence, by lemma 1,

$(q, x\varepsilon, \perp) \rightarrow^* (q, \varepsilon, [^0\perp) \rightarrow_3 (q, \varepsilon, \varepsilon)$.

As a result, x is accepted.

If x is not balanced, then either

(1) it is not prebalanced($\exists$ a prefix y of x, $L(y) < R(y)$) or

(2) x is prebalanced ($\forall$prefix y of x, $L(y) > R(y)$)

For the former case, by lemma 1, $(q, x, \perp)$ is blocked and x is not accepted.

For the latter case, by lemma 1, $(q, x, \perp) \rightarrow^* (q, \varepsilon, \alpha\perp)$ iff $\alpha = [^{L(x)-R(x) > 0}$ contains one or more [.

But then $(q, \varepsilon, \alpha\perp)$ is a dead configuration (which cannot move further) and is not accepted! Hence x is not accepted!

Another example

- The set $\{ww \mid w \in \{a,b\}^*\}$ is known to be not Context-free but its complement
  $L_1 = \{a,b\}^* - \{ww \mid w \in \{a,b\}^*\}$ is.

Exercise: Design a NPDA P2 to accept L_1 by empty stack.

Hint: $x \in L_1$ iff

(1) $|x|$ is odd or

(2) $x = yazybz'$ or $ybzyaz'$ for some $y,z,z' \in \{a,b\}^*$
with $|z|=|z'|$, which also means
$x = yay'ubu'$ or $yby'uau'$ for some $y,y',u,u' \in \{a,b\}^*$
with $|y|=|y'|$ and $|u|=|u'|$.

P2 behaves as follows: Nondeterministicallly guess input has odd or even length // (ε,⊥)→(q0, ⊥);(ε,⊥)→(q2,⊥); (ε,⊥)→(q6,⊥)

case odd length :

q0 : on any input c, goto q1 // (c, ⊥) → (q1, ⊥), c is 'a' or 'b'

q1: on any input c, go to q0 ; // (c, ⊥) → (q0, ⊥)

on (ε, ⊥) pop ⊥ (and accept). // (ε , ⊥) → (q1, ε)

case even length:

// q2~q5 : handle case: input = xayubv with |x|=|y| and |u|=|v|

q2: (c, s) → (q2, o s);(a, s) → (q3, s) // push o until 'a'

q3: (c, o) → (q3, ε) ; (c, ⊥) → (q4, ⊥) // pop o foreach c until ⊥

q4: (c, s) → (q4, o s) ; (b, s) → (q5, s) //push o foreach c until 'b'

q5: (c,o) →(g5, ε) ; // pop o foreach c unitl ⊥

(ε, ⊥) → (q5, ε) // pop ⊥ and accept

// q6 ~ q9 :handle case: input = xbyuav with |x|=|y| and |u|=|v|

… (left as an exercise)

More Examlpes

- **Find PDA for each of the following languages:**
  - L1=$\{w\in\{0,1\}^* \mid$ the length of w is odd and its middle symbol is 0.$\}$
  - L2=$\{w\in\{0,1\}^* \mid$ w contains as many 0s as 1s.$\}$
  - L3 =$\{w\in\{0,1\}^* \mid$ w contains more 1s than 0s.$\}$
  - L4 = $\{a^n b^m c^k \mid k = m + 2n\}$.
  - L5 = $\{w\in\{a,b,c\}^* \mid \#a(w) + \#b(w) \neq \#c(w)\}$.
    - // #a(w) is the number of a's occurring in w.
  - L6 = $\{w\in\{a,b\}^* \mid \#a(w) \leq 2 \times \#b(w)\}$.

Equivalent expressive power of both types of acceptance

- $M = (Q,\Sigma,\Gamma,\delta,s,,F)$: a PDA

  Let u, t : two new states $\notin Q$ and

  $\blacklozenge$: a new stack symbol $\notin \Gamma$.

- Define a new PDA $M' = (Q',\Sigma,\Gamma',\delta',s', \blacklozenge, F')$ where
  - $Q' = Q \cup \{u, t\}$, $\Gamma' = \Gamma \cup \{\blacklozenge\}$, $s' = u$, $F' = \{t\}$ and
  - $\delta' = \delta \cup \{ (u,\varepsilon, \blacklozenge) \dashrightarrow (s, \bot\blacklozenge) \}$ // push $\bot$ and call M
  - $\cup \{ (f, \varepsilon, A) \to (t,A) \mid f \in F \text{ and } A \in \Gamma' \}$ /* return to M'
  - after reaching final states */
  - $\cup \{(t, \varepsilon,A) \dashrightarrow (t,\varepsilon) \mid A \in \Gamma' \}$ // pop until EmptyStack

- Diagram form relating M and M': see next slide.

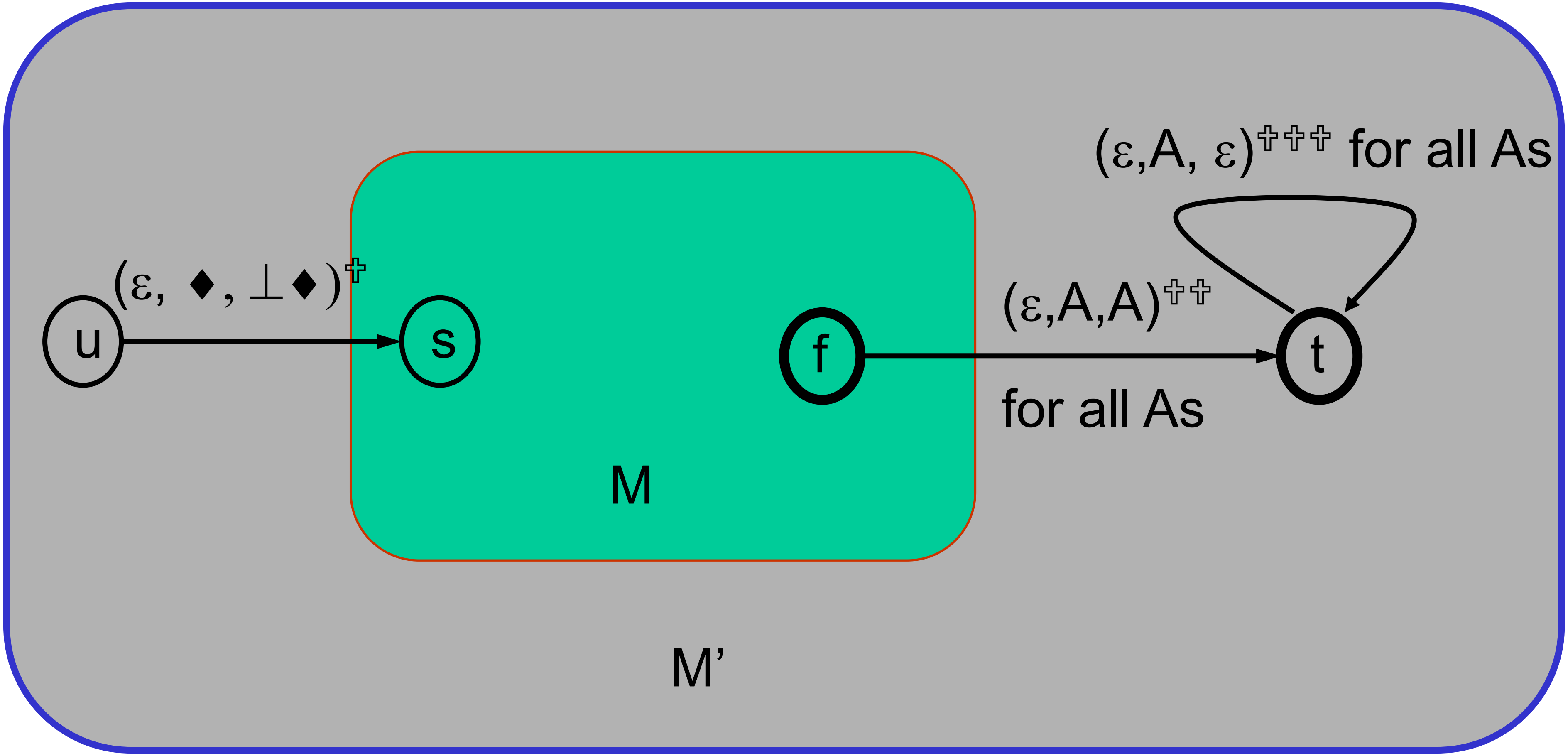
Theorem: $L_f(M) = L_e(M')$

pf: M accepts x $\Rightarrow (s, x, \bot) \dashrightarrow^n_M (q, \varepsilon, \gamma)$ for some $q \in F$

$\Rightarrow (u, x, \blacklozenge) \dashrightarrow_{M'} (s, x, \bot\blacklozenge) \dashrightarrow^n_{M'} (q, \varepsilon, \gamma\blacklozenge) \dashrightarrow_{M'} (t, \varepsilon, \gamma\blacklozenge)$

$\dashrightarrow^*_{M'} (t,\varepsilon, \varepsilon)$ $\Rightarrow$ M' accepts x by empty stack.

From final state to emptystack:



✞: push ⊥ and call M

✞✞ : return to t of M' once reaching final states of M

✞✞✞: pop all stack symbols until emptystack

From FinalState to EmptyStack

Conversely, M' accepts x by empty stack

$\Rightarrow$ $(u, x, \blacklozenge) \text{-->}_{M'} (s, x, \perp\blacklozenge) \text{-->}^*_{M'} (q, y, \gamma \blacklozenge) \text{-->} (t, y, \gamma\blacklozenge) \text{-->}^*$ $(t, \varepsilon, \varepsilon)$ for some $q \in F$

$\Rightarrow$ $y = \varepsilon$ since M' cannot consume any input symbol after it enters state t. => M accepts x by final state.

- **Define next new PDA M'' = $(Q', \Sigma, \Gamma', \delta'', s', \blacklozenge, F')$ where**
  - ▯ $Q' = Q \cup \{u, t\}$, $\Gamma' = \Gamma \cup \{\blacklozenge\}$, $s' = u$, $F' = \{t\}$ and
  - ▯ $\delta'' = \delta \cup \{ (u, \varepsilon, \blacklozenge) \text{-->} (s, \perp\blacklozenge) \}$ // push $\perp$ and call M
  - ▯ $\cup \{ (p, \varepsilon, \blacklozenge) \text{->} (t, \varepsilon) \mid p \in Q \}$ /* return to M'' and accept
  - ▯ if EmptyStack */
  - ▯
- **Diagram form relating M and M'': See slide 15.**

From EmptyStack to FinalState

- Theorem: $L_e(M) = L_f(M'')$.

pf: M accepts x => $(s, x, \perp) \to^n_M (q, \varepsilon, \varepsilon)$

=> $(u, x, \blacklozenge) \to_{M''} (s, x, \perp\blacklozenge) \to^n_{M''} (q, \varepsilon, \varepsilon\, \blacklozenge) \to_{M''} (t, \varepsilon, \varepsilon)$

=> M'' accepts x by final state (and empty stack).

Conversely, M'' accepts x by final state (and empty stack)

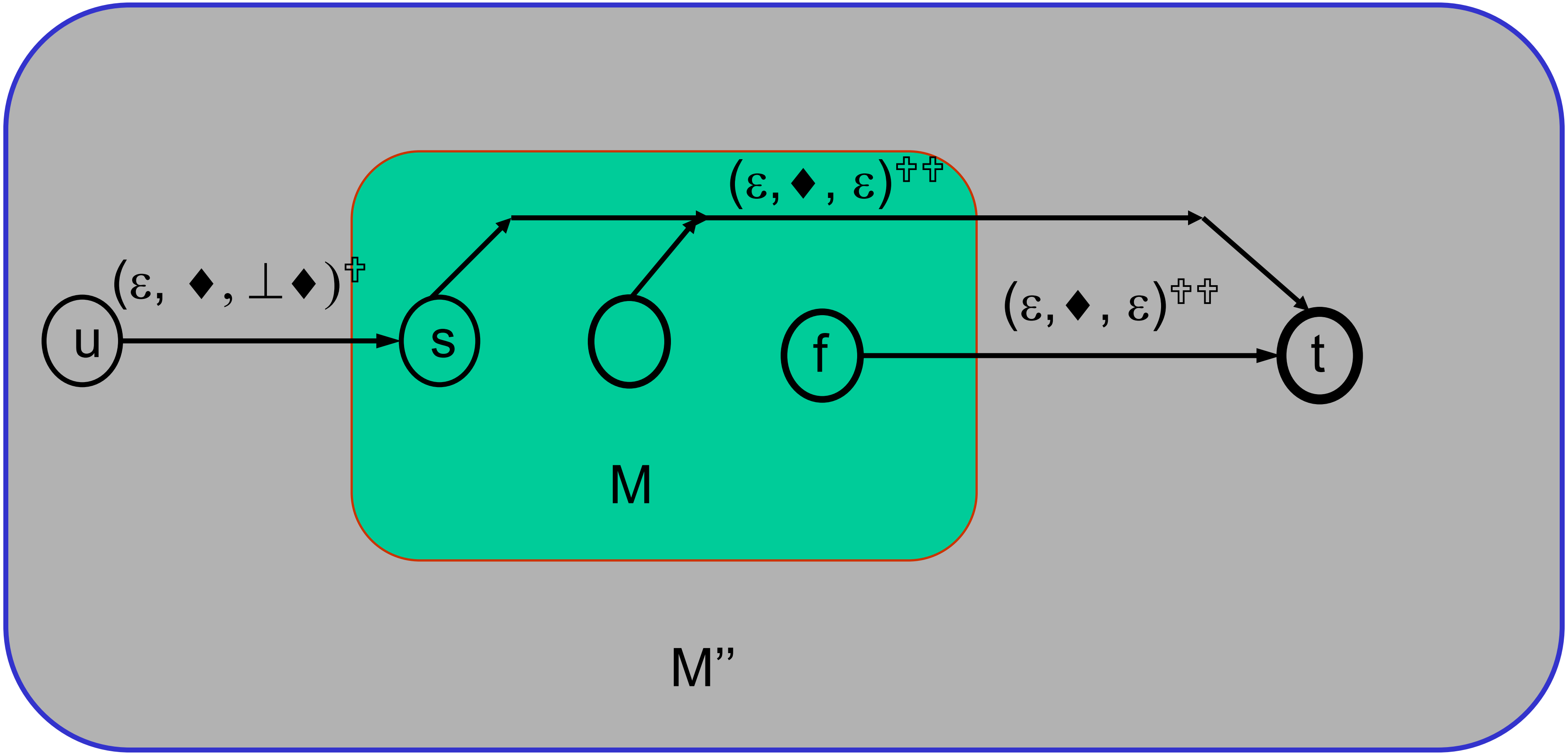
=> $(u, x, \blacklozenge) \to_{M''} (s, x, \perp\blacklozenge) \to^*_{M''} (q, y, \blacklozenge) \to_{M''} (t, \varepsilon, \varepsilon)$ for some state q in Q

=> $y = \varepsilon$ [and STACK= ε] since M'' does not consume any input symbol at the last transition $((q, \varepsilon, \blacklozenge), (t, \varepsilon))$

=> M accepts x by empty stack.

QED

From emptystack to final state (and emptystack)



✝ : push ⊥ and call M

✝✝: if emptystack (i.e.see ♦ on stack) , then pop ♦ and return to state t of M''

Equivalence of PDAs and CFGs

- Every CFL can be accepted by a PDA (with only one state).
- $G = (N, \Sigma, P, S)$: a CFG.
  - wlog assume all productions of G are of the form:
  - $A \to c\ B_1B_2B_3\ldots B_k$ ($k \geq 0$) and $c \in \Sigma \cup \{\varepsilon\}$.
  - note: 1. $A \to \varepsilon$ satisfies such constraint; 2. can require $k \leq 2$.
- Define a PDA $M = (\{q\}, \Sigma, N, \delta, q, S, \{\})$ from G where
  - q is the only state (hence also the start state),
  - Σ, the set of terminal symbols of G, is the input alphabet of M,
  - N, the set of nonterminals of G, is the stack alphabet of M,
  - S, the start nonterminal of G, is the initial stack symbol of M,
  - {} is the set of final states. (hence M accepts by empty stack!!)
  - $\delta = \{\ ((q,c,A), (q, B_1B_2\ldots B_k))\ \mid A \to c\ B_1B_2B_3\ldots B_k \in P\ \}$

Example

- G : 1. S -> [B S (q, [, S) --> (q, B S)
  2. S -> [B (q, [, S) --> (q, B)
  3. S-> [S B ==> δ : (q, [, S) --> (q, S B)
  4. S -> [S B S (q, [, S) --> (q, S B S)
  5. B ->] (q,], B) --> (q, ε)

- L(G) = the set of nonempty balanced parentheses.
- leftmost derivation v.s. computation sequence (see next table)

S $^L\text{-->}^*_G$ [[[ ]] [ ]] <==> (q, [[[]][]], S) -->^*_M (q, ε, ε)

S $^L\text{-->}^n_G$ [[[] BSB <==> (q, [[[]][]], S) -->^n_M (q,][]] , BSB)

A $^L\text{-->}^n_G$ z γ <==> (q, z y , A) -->^n_M (q, y , γ)

rule applied	sentential form of left-most derivation	configuration of the pda accepting x
	S	(q, [[[ ]] [ ]], S)
3	[S B	(q, [[[[ ]] [ ]], SB)
4	[[S B S B	(q, [[[ ]] [ ]], SBSB)
2	[[[B B S B	(q, [[[ ]] [ ]], BBSB)
5	[[[ ] B S B	(q, [[[ ]] [ ]], BSB)
5	[[[ ]] S B	(q, [[[ ]] [ ]], SB)
2	[[[ ]] [B B	(q, [[[ ]] [ ]], BB)
5	[[[ ]] [ ] B	(q, , [[[ ]] [ ]], B)
5	[[[[ ]] [ ]]	(q, , [[[ ]] [ ]] ,)

leftmost derivation v.s. computation sequence

Lemma 1: For any $z,y \in \Sigma^*$, $\gamma \in N^*$ and $A \in N$,

$$A \stackrel{L}{\longrightarrow}{}^{n}_{G}\ z\gamma \quad \text{iff} \quad (q, zy, A) \longrightarrow^{n}_{M} (q, y, \gamma)$$

Ex: $S \stackrel{L}{\longrightarrow}{}^{3}_{G}$ [[[BBSB <==> (q, [[[ ]][]] , S) $\longrightarrow^{3}_{M}$ (q,]][]], BBSB)

pf: By ind. on n.

Basis: n = 0. $A \stackrel{L}{\longrightarrow}{}^{0}_{G}\ z\gamma$ iff $z = \varepsilon$ and $\gamma = A$

iff $(q, zy, A) \longrightarrow^{0}_{M} (q,y,\gamma)$

Ind. case: 1. (only-if part)

Suppose $A \stackrel{L}{\longrightarrow}{}^{n+1}_{G}\ z\gamma$ and $B \to c\beta$ was the last rule applied.

I.e., $A \stackrel{L}{\longrightarrow}{}^{n}_{G}\ uB\alpha \stackrel{L}{\longrightarrow}{}_{G}\ uc\beta\alpha = z\gamma$ with $z = uc$ and $\gamma = \beta\alpha$.

Hence $(q, u\,cy, A) \longrightarrow^{n}_{M} (q, cy, B\alpha)$ // by ind. hyp.

$\longrightarrow_{M} (q, y, \beta\alpha)$ // since $((q,c,B),(q, \beta)) \in \delta$

leftmost derivation v.s. computation sequence (cont'd)

2. (if-part) Suppose $(q, zy, A) \text{-->}^{n+1}_M (q, y, \gamma)$ and $((q,c,B),(q, \beta)) \in \delta$ was the last transition executed. I.e.,

$(q, zy, A) = (q, ucy, A) \text{-->}^{n}_M (q, cy, B\alpha) \text{-->}_M (q, y, \beta\alpha) =(q,y, \gamma)$.

where $z = uc$ and $\gamma = \beta\alpha$ for some u, α. But then

$A \ {}^L\text{-->}^n_G \ uB\alpha$ // by ind. hyp.,

${}^L\text{-->} \ uc\,\beta\alpha = z\,\gamma$ // since by def. $B \text{->} c\,\beta \in P$

Hence $A \ {}^L\text{-->}^{n+1}_G \ z\,\gamma$ QED

Theorem 2: L(G) = L(M).

pf: $x \in L(G)$ iff $S \ {}^L\text{-->}^*_G \ x$

iff $(q, x, S) \text{-->}^*_M (q, \varepsilon, \varepsilon)$

iff $x \in L(M)$. QED

Simulating PDAs by CFGs

Claim: Every language accepted by a PDA can be generated by a CFG.

- **Proved in two steps:**
  - **1. Special case : Every PDA with only one state has an equivalent CFG**
  - **2. general case: Every PDA has an equivalent CFG.**

- **Corollary: Every PDA can be minimized to an equivalent PDA with only one state.**

pf: M : a PDA with more than one state.

1. apply step 2 to find an equivalent CFG G

2. apply theorem 2 on G , we find an equivalent PDA with only one state.

PDA with only one state has an equivalent CFG.

- $M = (\{s\}, \Sigma, \Gamma, \delta, s, \perp, \{\})$: a PDA with only one state.

  Define a CFG $G = (\Gamma, \Sigma, P, \perp)$ where

$$P = \{ A \to c\beta \mid ((q, c, A), (q, \beta)) \in \delta \}$$

Note: M ==> G is just the inverse of the transformation :

G ==> M defined at slide 22.

Theorem: L(G) = L(M).

Pf: Same as the proof of Lemma 1 and Theorem 2.

Simulating general PDAs by CFGs

- **How to simulate arbitrary PDA by CFG ?**
  - **idea: encode all state/stack information in nonterminals !!**

Wlog, assume M = (Q, Σ, Γ, δ, s, $\perp$, {t}) be a PDA with only one final state and M can empty its stack before it enters its final state. (The general pda M'' at slide 21 satisfies such constraint.)

Let $N \subseteq Q \times \Gamma^* \times Q$.

Elements of N such as (p, ABC, q) are written as <pABCq>.

Define a CFG G = (N, Σ, <s$\perp$t>, P) based on M, where

P = { <pAr> → c <q $\underline{B_1\ B_2 \ldots B_k}$ r>

| ((p,c,A), ($\boxed{q,\ B_1B_2\ldots B_k}$)) $\in \delta$, $k \geq 0$, c $\in \Sigma$ U {ε}, r $\in$ Q }

U // Rules for nonterminals <q B_1 B_2 …B_k r>

{ <pAαr> → <pAq><qαr>, <pp> → ε | p,q,r $\in$Q and $\alpha \in \Gamma^*$}

For a computation process :

$(p, wy, A_1A_2\ldots A_n\ \beta) \rightarrow^*_M\ (r, y, \beta)$,

there must exists $x_1x_2\ldots x_n = w$ **and states** $q_1,q_2,\ldots, q_n$ **such that**

$$(p, wy, A_1A_2\ldots A_n\beta)$$
$$\rightarrow^*_M\ (q_1, x_2\ldots x_ny, A_2\ldots A_n\beta)$$
$$\rightarrow^*_M\ (q_2, x_3\ldots x_ny, A_3\ldots A_n\beta) \rightarrow^* \ldots$$
$$\rightarrow^*_M\ (q_{n-1},\ x_ny, A_n\beta) \rightarrow^*_M\ (q_n = r,\ y,\beta).$$

We want the grammar derivation $\rightarrow_G$ **to simulate such computation:**

$$\langle pA_1A_2\ldots A_nr\rangle$$
$$\rightarrow^* x_1\ \langle q_1A_2\ldots A_nr\rangle$$
$$\rightarrow^* x_1x_2\ \langle q_2A_2\ldots A_nr\rangle \rightarrow^*\ \ldots$$
$$\rightarrow^* x_1x_2\ldots x_{n-1}\ \ \langle q_{n-1}A_nr\rangle\ \ \rightarrow^* x_1x_2\ldots x_n\ \langle q_nr\rangle = x_1x_2\ldots x_n\ \ \text{if } q_n = r$$

Rules for <p A_1 A_2 …A_n r>

case 1: n = 0 : (p, wy, $A_1A_2\ldots A_n$ β) $\rightarrow^*_M$ (r, y, β) .

I.e, (p, wy, β) $\rightarrow^*_M$ (r, y, β) ,

which must be permitted if w = ε and p = r.

we thus have rule <pp> → ε for all state p.

case 2: n > 1 : (p, wy, $A_1A_2\ldots A_n$ β) $\rightarrow^*_M$ (r, y, β) . Then there must exists uv = w, and state q such that

(p, u, A_1a) →*M (q, ε, α) and

(p, wy, $A_1A_2\ldots A_n$ β) $\rightarrow^*_M$ (q, vy, $A_2\ldots$Anβ) $\rightarrow^*_M$ (r, y, β)

We thus have the assumption:

<pAq> →* u and <q $A_2\ldots A_n$r> → v.

and the rules <p$A_1A_2\ldots A_n$ r> → <pAq><q$A_1A_2\ldots A_n$r> for all states p,q,r.

Rules for <p A_1 A_2 ...A_n r>

case 3: n = 1 : $(p, wy, A\ \beta) \rightarrow^*_M (r, y, \beta)$.

This is possible only if

$(p, wy, A\ \beta) \rightarrow_M (q, vy, \gamma\beta) \rightarrow^*_M (r, y, \beta)$.,

where w = cv and ((p,c,A), (q, r)) is an instruction.

We thus need rule $<pAr> \rightarrow c<q\gamma r>$ for all states r,

and the assumption: $<q\gamma r> \rightarrow^* v$,

to guarantee the derivation:

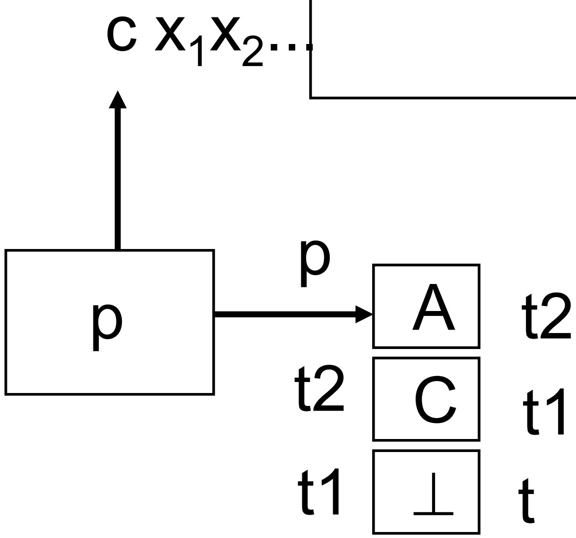
$<pAr> \rightarrow c<q\gamma r> \rightarrow^* cv = w$.

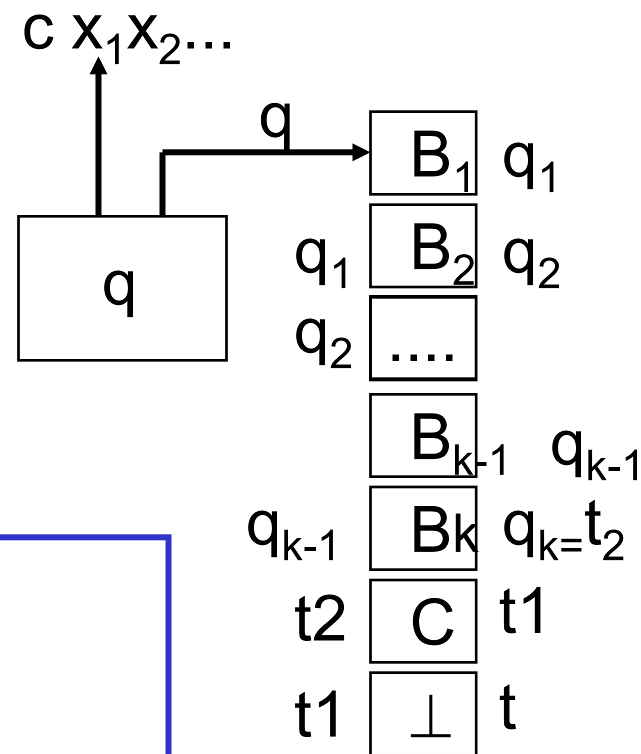
machine view: $(p, c, A) \rightarrow (q, B_1B_2...B_k)$

rule views : <pAr> → c<q $B_1B_2...B_k$ r> for each r ∈Q

<pABCr> → <pAq><qBCr> for each q ∈Q

<pεp> → ε <pεq> → no rule if p ≠q





We want to use derivation <pαq> →* w
to simulate the computation:

$(p, wy, \alpha\beta) \rightarrow^*_M (q, y, \varepsilon\beta)$

So, if $(p,c,A) \rightarrow_M (q, \beta)$ we have rules :
<p A r> → c <q β r> for all states r.

Simulating PDAs by CFG (cont'd)

- Note: Besides storing sate information on the nonterminals, G simulate M by guessing nondeterministically what states M will enter at certain future points in the computation, saving its guesses on the sentential form, and then verifying later that those guesses are correct.

Lemma 25.1: if $<pB_1B_2\ldots B_kq>$ is a nonterminal, then

$$(p,x,B_1B_2\ldots B_k) \dashrightarrow^*_M (q,\varepsilon,\varepsilon) \quad \text{iff}$$

$$<pB_1B_2\ldots B_kq> \rightarrow^*_G x. \qquad (*)$$

Notes: 1. when k = 0 (*) is reduced to $<pq> \rightarrow^*_G x$, where $<pq> = \varepsilon$ if p=q and $<pq>$ is undefined if $p \neq q$.

2. In particular, $(p,x,B) \dashrightarrow^*_M (q,\varepsilon,\varepsilon)$ iff $<pBq> \rightarrow^*_G x$.

Pf: by ind. on n. Basis: k = 0.

LHS holds iff ($x = \varepsilon$, k = 0, and p = q) iff RHS holds.

Simulating PDAs by CFGs (cont'd)

Inductive case:

(=>:) Suppose $(p,x,B_1B_2\ldots B_k) \text{-->}^*_M (q,\varepsilon,\varepsilon)$ and $((p,c,B_1),(r,C_1C_2\ldots C_m))$ is the first instr. executed. I.e.,

$(p,x,B_1B_2\ldots B_k) \text{-->}_M (r, y, C_1C_2\ldots C_mB_2\ldots B_k)$

$\rightarrow^*_M (s, z, B_2\ldots B_k)$

$\text{-->}^*_M (q,\varepsilon,\varepsilon)$, where $x = cy = cdz$.

By ind. hyp.,

$\langle rC_1C_2\ldots C_ms\rangle \rightarrow^* d$ since $(r, d\ C_1C_2\ldots C_m) \text{-->}^* (s,\varepsilon,\varepsilon)$ and

$\langle sB_1\ldots B_kq\rangle \rightarrow^* z$

Hence $\langle pB_1B_2\ldots B_kq\rangle \rightarrow \langle pB_1r\rangle\langle rB_1B_2\ldots B_kq\rangle$

$\rightarrow c\ \langle rC_1C_2\ldots C_ms\rangle\ \langle rB_1B_2\ldots B_kq\rangle \rightarrow^* cdz = x$

Simulating PDAs by CFGs (cont'd)

(<=:) $\langle pB_1B_2\ldots B_kq\rangle \ {}^{L}\!\!\rightarrow^*_G x$.

Suppose $\langle pB_1B_2\ldots B_kq\rangle \rightarrow \langle p\ B_1\ q_1\rangle \langle q_1B_2\ldots B_kq\rangle$

${}^{L}\!\!\rightarrow_G c \langle r_0\ C_1C_2 \ldots C_m\ q_1\rangle \langle q_1\ B_2\ldots B_kq\rangle$

${}^{L}\!\!\rightarrow_G^n\ cy \quad (= x\ )$

where $\langle pB_1q_1\rangle \rightarrow c \langle r_0\ C_1C_2\ldots C_m\ q_1\rangle \in P$ --(*) .

But then since, by (*), $[(p, c, B1)\ , (r_0, C_1C_2\ldots C_m)]$ – (**) is an instr of M,

$(p,x,B_1\ldots B_k) \text{-->}_M (r_0, y, C_1C_2\ldots C_mB_2\ldots B_n)$ --- By (**)

$\text{-->}^* (q1, z, B_2\ldots B_n)$ -- by IH

$\text{-->}^n{}_M (q,\varepsilon,\varepsilon)$. -- ,by ind. hyp. QED

Theorem 25.2 L(G) = L(M).

Pf: $x \in L(G)$ iff $\langle s\bot t\rangle \rightarrow^* x$

iff $(s,x,\bot) \text{-->}^*_M (t,\varepsilon,\varepsilon)$ ---- Lemma 25.1

iff $x \in L(M)$. QED

Example

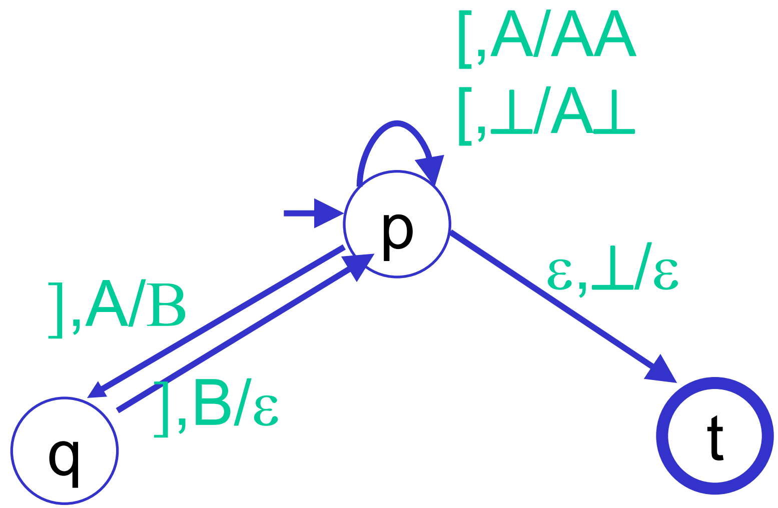
- L = {x∈ {[,]}* | x is a balanced string of [and]], i.e., #](x) = 2 #[(x) and all “]]“s must occur in pairs }
- Ex: [ ]] [[ ]]]] ∈ L but [ ] [ ]]] ∉ L.
- L can be accepted by the PDA

M = (Q, Σ, Γ, δ, p, ⊥,{t}), where

Q = {p,q,t}, Σ = {[,]}, Γ = {A, B, ⊥},

and δ is given as follows:

- (p, [, ⊥) --> (p, A⊥),
- (p,[,A) --> (p,AA),
- (p,], A) --> (q, B),
- (q,], B) --> (p, ε),
- (p,ε, ⊥) --> (t,ε)



- M can be simulated by the CFG G = (N,Σ, <p⊥t>, P) where
  - N = { <X D Y> | X,Y ∈{p,q,t } and D ∈ { A,B,⊥}* },
  - and P is derived from the following pseudo rules :
  - (p, [, ⊥) --> (p, A⊥) : <p⊥?> → [<pA⊥?>
  - (p,[,A) --> (p,AA) : <p A ?> → [<pAA?>
  - (p,], A) --> (q, B), : <p A ?> →] <qB?>
  - Each of the above produces 3 rules (? = p or q or t).
  - (q,], B) --> (p, ε), : <q B ?> →] <p ε ?>
  - This produces only 1 rule : <qBp> →]
  - (? = p, but could not be q or t why ?)
  - <q B ?> →] <p ε ?> => <qBp> →] <pεp> $\rightarrow^0$]
  - (p,ε, ⊥) --> (t,ε) : <p⊥?> → <t ε ?>
  - This results in one rule : <p⊥t> → ε

- $\langle p \perp ?\rangle \to [\ \langle pA\perp ?\rangle$ ➔ results in 3 rules : ? = p, q or t.
- $\langle p \perp p\rangle \to [\ \langle pA\perp p\rangle$ ---(1)
- $\langle p \perp q\rangle \to [\ \langle pA\perp q\rangle$ ---(2)
- $\langle p \perp t\rangle \to [\ \langle pA\perp t\rangle$ ---(3)
- (1)~(3) each again need to be expanded into 3 rules.
- $\langle pA\perp p\rangle \to \langle pA?\rangle\langle ? \perp p\rangle$ where ? is p or q or t.
  - ✪ $\langle pA\perp p\rangle \to \langle pAp\rangle\langle p \perp p\rangle$
  - ✪ $\langle pA\perp p\rangle \to \langle pAq\rangle\langle q \perp p\rangle$
  - ✪ $\langle pA\perp p\rangle \to \langle pAt\rangle\langle t \perp p\rangle$
- $\langle pA\perp q\rangle \to \langle pA?\rangle\langle ? \perp q\rangle$ where ? is p or q or t.
  - ✪ $\langle pA\perp q\rangle \to \langle pAp\rangle\langle p \perp q\rangle$
  - ✪ $\langle pA\perp q\rangle \to \langle pAq\rangle\langle q \perp q\rangle$
  - ✪ $\langle pA\perp q\rangle \to \langle pAt\rangle\langle t \perp q\rangle$
- $\langle pA\perp t\rangle \to \langle pA?\rangle\langle ? \perp t\rangle$ where ? is p or q or t.
  - ✪ …

- Similarly <pA?> → [<pAA?> results in 9 rules:
- Where $?_2$ = p,q, or t.
- <p A p> →[<pA$?_2$> <$?_2$⊥p> ---(1)
  - <p A p> → [<pAp> <p⊥p>
  - <p A p> → [<pAq> <q⊥p>
  - <p A p> → [<pAt> <t⊥p>
- <p A q> →[<pA$?_2$> <$?_2$⊥q> ---(2)
  - …
- <p A t> → [<pA$?_2$> <$?_2$⊥t> ---(3)
  - …

Problem: How many rules are there in the generated grammar ?

- Let m be the max number of symbols pushed in δ.
  - i.e., $m = \max \{ |\beta| \mid (p,c,A) \rightarrow (q, \beta) \in \delta \}$
  - Then $|G| = O(|\delta| \times |Q|^m)$ --- (1)
- Notes:
  - 1. $|Q| = 10$, $m = 3 \Rightarrow |G| = 1000 |\delta|$
  - 2. Each instruction $(p,c,A) \rightarrow (q, B_1\ldots B_m)$ induces the $|Q|^m$ rules
  - $\{<pAX_m> \rightarrow c<qB_1X_1><X_1B_2X_2>\ldots<X_{m-1}B_mX_m> \mid X_1,X_2,\ldots X_m \in Q \}$

3. If m = 2 or use intermediate symbols/rules: Then

- $|G| = O(|\delta| x |Q| x m \times |Q|)$. Instr $(p,c,A) \rightarrow (q,B_1\ldots B_m)$ induces rules
- **{** $\quad\quad <pAX_m> \rightarrow c <qB_1B_2B_3\ldots B_mX_m>,$
- $<qB_1B_2B_3\ldots B_mX_m> \rightarrow <qB_1X_1> <X_1B_2B_3\ldots B_mX_m>,$
- $<X_1B_2B_3\ldots B_mX_m> \rightarrow <X_1B_2X_2><X_2B_3\ldots B_mX_m>, \ \ldots$
- $<X_{m-2}B_{m-1}B_mX_m> \rightarrow <X_{m-2}B_{m-2}X_{m-1}><X_{m-1}B_mX_m>$
- $\mid X_1,X_2,\ldots X_m \in Q \}$ **// mx|Q| nonterminals** $<X_JB_{J+1}B_3\ldots B_mX_m>,$