Pushdown Automata and Context-Free Languages

NPDAs

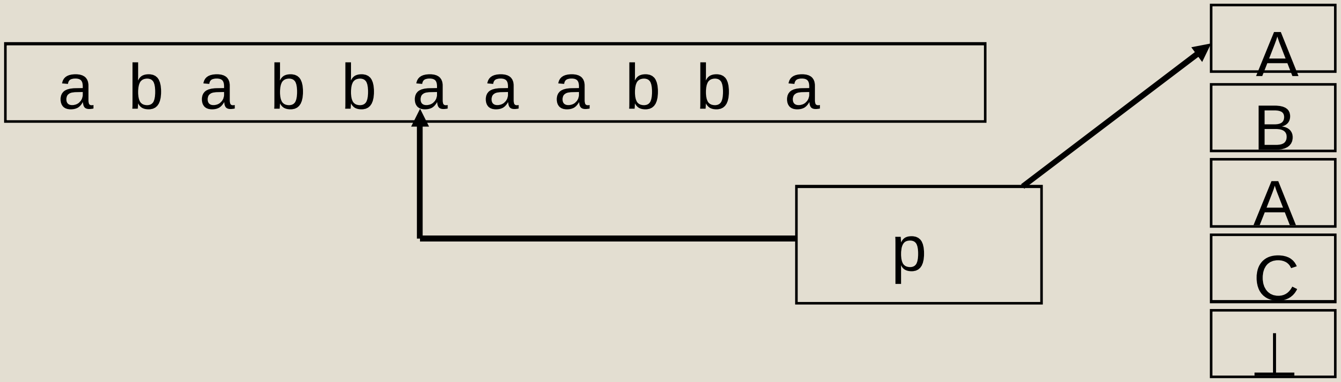
- A NPDA (Nondeterministic PushDown Automata) is a 7-tuple
  M = (Q,S,G, d ,s, $\perp$, F) where
  - Q is a finite set (the states)
  - S is a finite set (the input alphabet)
  - G is a finite set (the stack alphabet)
  - d $\subseteq$ (Q x (S U {e})x G) x (Q x G*) is the transition relation
  - s $\in$ Q is the start state
  - $\perp \in$ G is the initial stack symbol
  - F $\subseteq$ Q is the final or accept states
- $((p,a,A),(q,B_1B_2...B_k)) \in d$ 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 $B_1B_2...B_k$ 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.

$((p,e,A),(q,B_1B_2...B_k)) \in d$ means similar to $((p,a,A),(q,B_1B_2...B_k)) \in d$ 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 x S* x G*.
- 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.
- $CF_M =_{def} Q \times S^* \times G^*$ is the set of all possible configurations for a NPDA M.
- One-step computation ($-->_M$) of a NPDA:
  - (p, ay, Ab) $-->_M$ (q, y , gb) for each ((p,a,A), (q, g)) $\in$ d.   (1)
  - (p, y, Ab) $-->_M$ (q, y, gb) for each ((p,e,A),(q, g)) $\in$ d . (2)
  - Let the next configuration relation $-->_M$ on CF_M^2 be the set of pairs of configurations satisfying (1) and (2).
  - $-->_M$ 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.,
  - $C -->^0_M D$ iff $C = D$.
  - $C -->^{n+1}_M$ iff $\exists E$ $C -->^n_M E$ and $E -->_M D$.
  - $C -->^*_M D$ iff $\exists n \geq 0$ $C -->^n_M D$.
  - 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
  - $(s,x, \perp) -->^*_M (p,e, a)$ for some final state $p \in F$.
  - 2. by empty stack: M accepts x by empty stack iff
  - $(s,x, \perp) -->^*_M (p,e, e)$ for any state p.
  - Remark: both kinds of acceptance have the same expressive power.

Language accepted by a NPDAs

M = (Q,S,G,d,s,,F) : a NPDA.

The languages accepted by M is defined as follows:

- 1. accepted by final state:
- $L_f(M) = \{x \mid M \text{ accepts } x \text{ by final state}\}$
- 2. accepted by empty stack:
- $L_e(M) = \{x \mid M \text{ accepts } x \text{ by empty stack}\}$.
- 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 : M_1: A NPDA accepting the set of balanced strings of parentheses [ ] by empty stack.

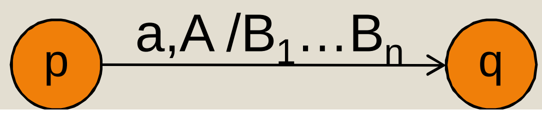
- M_1 requires only one state q and behaves as follows:

1. while input is '[' : push '[' onto the stack ;
2. while input is ']' and top is '[' : pop
3. while input is 'e' and top is $\perp$: pop.

Formal definition: Q = {q}, S = {[,]}, G = {[, $\perp$ },
start state = q, initial stack symbol = $\perp$.
d = { ((q,[, $\perp$), (q, [$\perp$)), ((q,[, [), (q, [[)), // 1
((q,], [), (q, e)), // 2
((q,e, $\perp$), (q, e)) } // 3

Transition Diagram representation of the program d :
((p,a A) , (q,B_1...B_n))$\in$ d =>

- 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
  $\rightarrow_M$ (q, [[ ]] [ ]] [ ], [⊥) instruction or transition (i)
  $\rightarrow_M$ (q, [ ]] [ ]] [ ], [[⊥) transition (ii)
  $\rightarrow_M$ (q,]] [ ]] [ ], [[[⊥) transition (ii)
  $\rightarrow_M$ (q,] [ ]] [ ], [[⊥) transition (iii)
  $\rightarrow_M$ (q, [ ]] [ ], [⊥) transition (iii)
  $\rightarrow_M$ (q,]] [ ], [[⊥) transition (ii)
  $\rightarrow_M$ (q,] [ ], [⊥) transition (iii)
  $\rightarrow_M$ (q, [ ], ⊥) transition (iii)
  $\rightarrow_M$ (q,], [⊥) transition (i)
  $\rightarrow_M$ (q, , ⊥) transition (iii)
  $\rightarrow_M$ (q, ,) transition (iv)
  accepts by empty stack

Failure computation of M1 on x

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

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

--> $*_M$ (q, [ ], $\perp$)

--> $_M$ (q, [ ],) transition (iv)

a dead state at 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., \$ D = (_, e, e) with empty input and stack s.t. (s,x,$\perp$) --> $*_M$ 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, a) is said to be blocked if the pda M cannot use up input z, i.e., there is no state r and stack b such that $(q, z, a) \rightarrow^* (r, e, b)$.
- Facts:
  - 1. If initial configuration $(s, z, \perp)$ is blocked then z is not accepted by M.
  - 2. If (q, z, a) is blocked then (q, zw, a) is blocked for all $w \in S^*$.

Pf: 1. If $(s, z, \perp)$ is blocked, then there is no state p, stack b such that $(s, z, \perp) \text{-->}^* (p, e, b)$, and hence z Is not accepted.

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

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

Pf: By induction on z.

basic case: z = e. Then $(q, zx, \perp) = (q, x, \perp) \to^0 (q, x, a\perp)$ iff $a = [^{L(z)-R(z)}$.

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

case 1: z = y[.

If y is prebalanced, then so is z. By ind. hyp. $(q, zx, \perp) = (q, y[, \perp) \to^* (q, [x, [^{L(y)-R(y)}\perp) \to (q, x, [[^{L(y)-R(y)}\perp) = (q, x, [^{L(z)-R(z)}\perp)$.

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].

If y is not prebalanced, then neither is z. By ind. hyp. $(q, y, \perp)$ is blocked, hence $(q, y], \perp)$ is blocked

If y is prebalanced and L(y) = R(y). Then z is not prebalanced. By ind. hyp., if $(q, y], \perp) \to^* (q,], a\perp)$ then $a = [^{L(z)-R(z)} = e$, but then $(q,], \perp)$ is blocked. Hence $(q, z, \perp)$ is blocked.

Finally, if y is prebalanced and L(y) > R(y). Then z is prebalanced, and

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

$\quad\quad --> (q, x, [^{L(y)-R(y)-1} \perp)$ --- (iii)

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

On the other hand, if

$(q,y]x,\perp) \dashrightarrow^* (q,x, a\perp)$ .Then there must exist a cfg $(q,]x, b)$ such that

$(q,y]x,\perp) \dashrightarrow^* (q,]x, b) \dashrightarrow^* (q,x, a\perp)$.

But then the intructions executed in the last part must be IV* III IV*.

If $(q,]x, b) \dashrightarrow_{IV^*\ III\ IV^*} (q,x, a\perp)$, then $b = \perp^m [\perp^n a \perp$. But by ind. hyp., $b = [^{L(y)-R(y)} \perp$, hence m = 0, n = 0 and $a = [^{L(y)-R(y)-1} \perp$.

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, $(q, xe,\perp) \dashrightarrow^* (q, e, [^0\perp) \dashrightarrow_{IV} (q, e, e)$. As a result, x is accepted.

If x is not balanced, it is not prebalanced. Hence, by lemma 1, $(q, x,\perp)$ is blocked and 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 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 $|u|=|u'|$.

Equivalent expressive power of both types of acceptance

- M = (Q,S,G,d,s,,F) : a PDA
  Let u, t : two new states $\notin$ Q and
  ♦ : a new stack symbol $\notin$ G.
- Define a new PDA M' = (Q',S,G',d',s', ♦, F') where
  - Q' = Q U {u, t}, G' = G U {♦ }, s' = u, F' = {t} and
  - d' = d U { (u,e, ♦) --> (s, ⊥♦) } // push ⊥ and call M
  - U { (f, e, A) -> (t,A) | f $\in$ F and A $\in$ G' } /* return to M'
  - after reaching final states */
  - U {(t, e,A) --> (t,e) | A $\in$ G' } // pop until EmptyStack
- Diagram form relating M and M': see next slide.

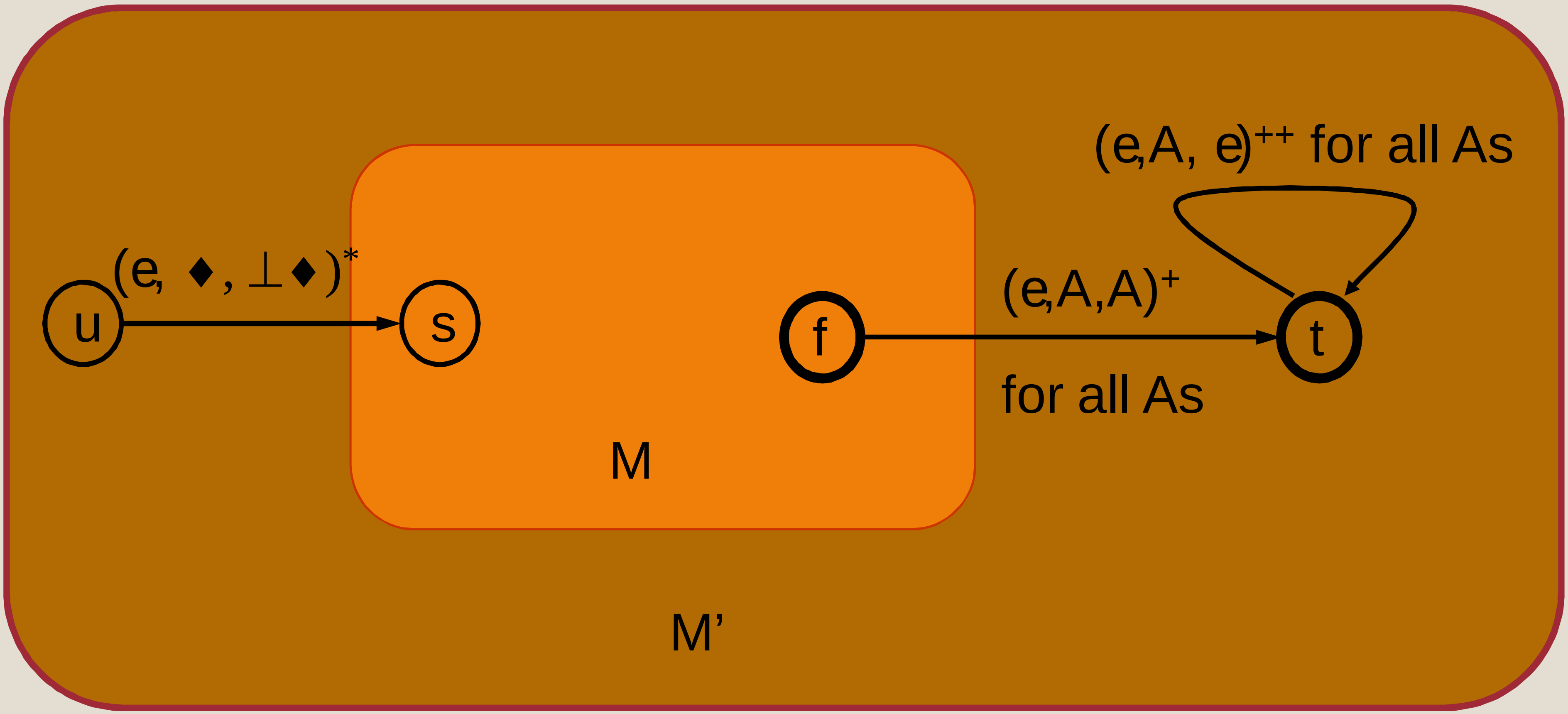
Theorem: $L_f(M) = L_e(M')$

pf: M accepts x => (s, x, ⊥) -->^n_M (q, e , g) for some q $\in$ F

=> (u, x, ♦) $\text{-->}_{M'}$ (s, x, ⊥♦) $\text{-->}^n_{M'}$ (q, e , g♦) $\text{-->}_{M'}$ (t, e , g ♦)

$\text{-->}^*_{M'}$ (t,e, e) => M' accepts x by empty stack.

From final state to empty stack:



*: 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

=> (u, x, ♦) $\text{-->}_{M'}$ (s, x, ⊥♦) $\text{-->}^*_{M'}$ (q, y, g ♦) --> (t, y, g♦) -->*

(t, e , e) for some q ∈ F

⇒ y = e since M' cannot consume any input symbol after it enters state t. => M accepts x by final state.

- Define next new PDA M'' = (Q',S,G',d'',s', ♦, F') where
  - Q' = Q U { u, t}, G' = G U {♦}, s' = u, F' = {t} and
  - d'' = d U { (u,e, ♦) --> (s, ⊥♦) } // push ⊥ and call M
  - U { (p,e,♦) -> (t, e) | p ∈ 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 => $(s, x, \perp) \longrightarrow^n_M (q, e, e)$

=> $(u, x, \blacklozenge) \longrightarrow_{M''} (s, x, \perp\blacklozenge) \longrightarrow^n_{M''} (q, e, e\blacklozenge) \longrightarrow_{M''} (t, e,$

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

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

=> $(u, x, \blacklozenge) \longrightarrow_{M''} (s, x, \perp\blacklozenge) \longrightarrow^*_{M''} (q, y, \blacklozenge) \longrightarrow_{M''} (t, e, e)$ f

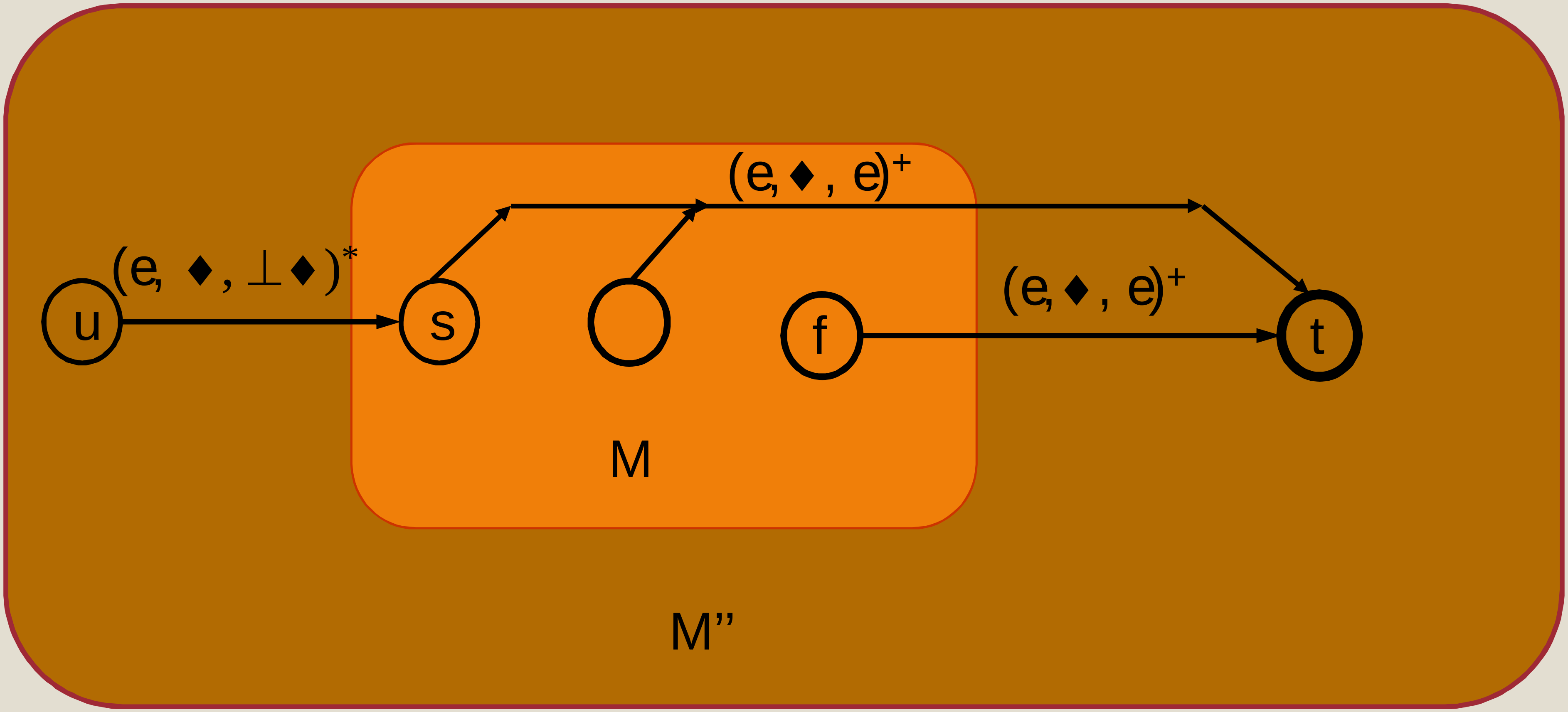
some state q in Q

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

=> 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.
- G = (N, S ,P,S) : a CFG.
  - wlog assume all productions of G are of the form:
  - $A \text{ -> } c\ B_1B_2B_3\ldots B_k$ ($k \geq 0$) and $c \in S \cup \{e\}$.
  - note: 1. A -> e satisfies such constraint; 2. can require $k \leq 2$.
- Define a PDA M = ({q}, S, N, d, q, S, { }) from G where
  - q is the only state (hence also the start state),
  - S, 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!!)
  - $d = \{\ ((q,c,A),\ (q,\ B_1B_2\ldots B_k))\ \mid A \text{ -> } c\ B_1B_2B_3\ldots B_k \in P\ \}$

Example

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

- L(G) = the set of nonempty balanced parentheses.

- leftmost derivation v.s. computation sequence
  (see next table)

$S \ {}^{L}\!\!\dashrightarrow^{*}_{G}$ [[[ ]] [ ]] <==> (q, [[[]][]], S) $\dashrightarrow^{*}_{M}$ (q, e, e)

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)
	[[[ ]] [ ] B	(q, , [[[ ]] [ ]], B)
5	[[[[ ]] [ ]]	(q, , [[[ ]] [ ]] ,)

leftmost derivation v.s. computation sequence

Lemma 24.1: For any z,y $\in$ S*, g $\in$ N* and A $\in$ N,

$A \ {}^{L}\text{-->}^{n}_{G} \ z\,g$ iff $(q, zy, A) \text{-->}^{n}_{M} (q, y, g)$

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

pf: By ind. on n.

Basis: n = 0. $A \ {}^{L}\text{-->}^{0}_{G} \ z\,g$ iff z = e and g = A

iff (q, zy, A) = (q, y, g) iff $(q, zy, A) \text{-->}^{0}_{M} (q,y,g)$

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

Suppose $A \ {}^{L}\text{-->}^{n+1}_{G} \ z\,g$ and B -> cb was the last rule applied.

I.e., $A \ {}^{L}\text{-->}^{n}_{G} \ uBa \ {}^{L}\text{-->}_{G}$ uc ba = z g with z = uc and g = ba

Hence $(q, u\,cy, A) \text{-->}^{n}_{M} (q, cy, Ba)$ // by ind. hyp.

$\text{-->}_{M} (q, y, ba)$ // since ((q,c,B),(q, b))

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

2. (if-part) Suppose (q, zy, A) -->^{n+1}_M (q, y, g) and ((q,c,B),(q, b)) $\in$ d is the last transition executed. I.e.,

(q, zy, A) -->^{n}_M (q, cy, Ba) -->_M (q, y, ba) with g = band z =
for some u. But the

A $^{L}\text{-->}^{n}_G$ uBa // by ind. hyp.,

$^{L}\text{-->}$ uc ba = z g // since by def. B -> c b $\in$ P

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

Theorem 24.2: L(G) = L(M).

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

iff (q, x, S) -->^{*}_M (q, e,)e

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 24.2 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

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

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

G ==> M defined at slide 16.

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

Pf: Same as the proof of Lemma 24.1 and Theorem 24.2.

Simulating general PDAs by CFGs

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

Wlog, assume M = (Q, S, G, d, s, ⊥, {t}) be a PDA with only one final state and M can empty its stack before it enters its final state. (The general pda at slide 15 satisfies such constraint.)

Let N ⊆ Q x G* x Q . Elements of N are written as <pABCq>.

Define a CFG G = (N, S, <s⊥t>, P) where

P = { <pAr> → c <q B_1 B_2 ...B_k r>

| ((p,c,A), (q, $B_1B_2...B_k$)) ∈ d, k ≥ 0, c ∈ S U {e

r ∈ Q }

Rules for $<q\ B_1\ B_2\ \ldots B_k\ r>$

We want $<qB_1 \ldots B_k\ r>$ to simulate the computation process in PDA M:

$(q, wy, B_1B_2\ldots B_k b) \mid\!- \ldots \mid\!- (r, y, b)$ iff $<qB_1\ldots B_k r> \rightarrow^* w$.

Hence: if k = 0. ie., $<qB_1B_2\ldots B_k r> = <q\varepsilon r>$, we should have

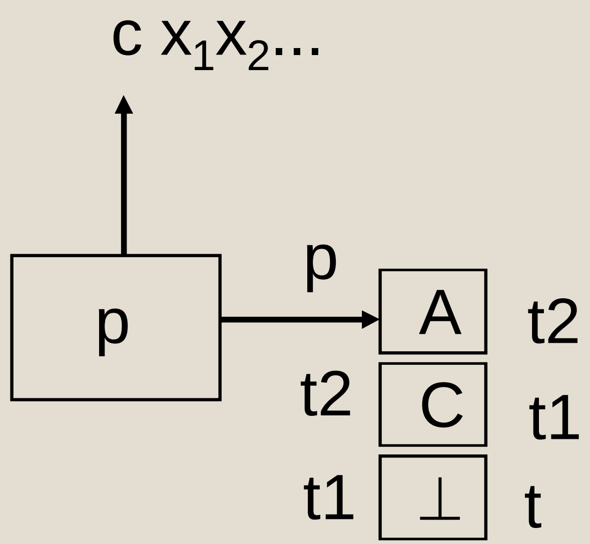
$<qr> \rightarrow e$ if q = r and

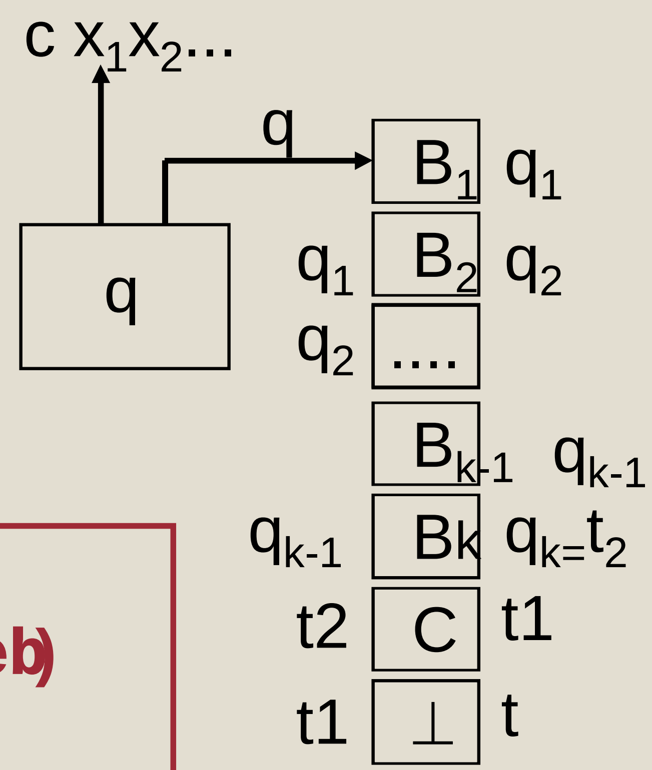
$<qr>$ has no rule if $q \neq r$.

If k > 1. Let $B_1B_2\ldots B_k = B_1D_2$, then :

- $<qB_1D_2r> \rightarrow S_{u1\in Q} <qB_1u_1> <u_1D_2r>$
- $\rightarrow S_{u1\in Q}\ S_{u2\in Q} <qB_1u_1> <u_1B_2u_2> <u_2D_2r>$
- $\rightarrow \ldots$
- $\rightarrow S_{u1\in Q}\ S_{u2\in Q} \ldots <qB_1u_1><u_1B_2u_2>\ldots<u_{k-1}B_kU_k><U_kD_kr>$
- $\rightarrow S_{u1\in Q}\ S_{u2\in Q} \ldots <qB_1u_1><u_1B_2u_2>\ldots<u_{k-1}B_k\ r>$

$(p, c, A) \dashrightarrow (q, B_1B_2\ldots B_k)$





We want to use <pAq> $\rightarrow^*$ w to simulate the computation: (p, wy, Ab) $\rightarrow^*_M$ (q, y, eb)

So, if (p,c,A) $\rightarrow_M$ (q, a) we have rules :

<p A r> $\rightarrow$ c <q a r> for all states r.

How to derive the rule **<p A r> → c <q a r>** ?

How to derive rules for the nonterminal : <q a r>

- case 1: $a = B_1B_2B_3 \ldots B_n$ (n > 0)
  - $=> <q\ a\ r> = <q\ B_1Q\ B_2QB_3Q \ldots QB_nr>$
  - $=> <q\ a\ r> \rightarrow <q\ B_1\ q_1> <q_1\ B_2\ q_2> \ldots$
  - $<q_{n-1}\ B_n\ r>$ for all states $q_1,q_2,\ldots,q_{n-1}$ in Q.
- case2: a = e.
  - q = r => <q a r> = <q e r> → e.
  - q != r => <q e r > cannot derive any string.

  - Then <pAq> → c <qeq> = c.

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 th guesses are correct.

Lemma 25.1: $(p,x,B_1B_2\ldots B_k) \text{-->}^n_M (q,e,e)$ iff

$\$\ q_1,q_2,\ldots q_k$ (=q) such that

$$\langle pB_1q_1\rangle\langle q_1B_2q_2\rangle\ldots\langle q_{k-1}B_kq\rangle \llcorner\!\rightarrow^n_G x. \quad (*)$$

Note: 1. when k = 0 (*) is reduced to $\langle pq\rangle \llcorner\!\rightarrow^n_G x$

2. In particular, $(p,x,B) \text{-->}^n_M (q,e,e)$ iff $\langle pBq\rangle \llcorner\!\rightarrow^n_G x$.

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

LHS holds iff (x = e, k = 0, and p = q) iff RHS holds.

Simulating PDAs by single-state PDAs (cont'd)

Inductive case:

(=>:) Suppose $(p,x,B_1B_2...B_k) \dashrightarrow^{n+1}_M (q,\varepsilon,\varepsilon)$ and

$\underline{((p,c,B_1),(r,C_1C_2...C_m))}$ is the first instr. executed. I.e.,

$(p,x,B_1B_2...B_k) \dashrightarrow_M (r, y, C_1C_2...C_mB_2...B_k) \dashrightarrow^n_M (q,\varepsilon,\varepsilon)$,

where x = cy.

By ind. hyp., ∃ states $r_1, \ldots, r_{m-1}, (r_m = q_1), q_2, \ldots q_{k-1}$ with

$<rC_1r_1><r_1C_2r_2>\ldots<r_{m-1}C_mq_1><q_1B_2q_2>\ldots<q_{k-1}B_kq_k> \Rightarrow^n_G y$

Also by the definition of G:

$\underline{<pB_1q_1> \rightarrow c <r_0C_1r_1><r_1C_2r_2>\ldots<r_{m-1}C_mq_1>}$ is a rule of G.

Combining both, we get:

$<pB_1q_1> <q_1B_2q_2> \ldots<q_{k-1}B_kq_k>$

$\Rightarrow_G c <r_0C_1r_1><r_1C_2r_2>\ldots<r_{m-1}C_mq_1> <q_1B_2q_2> \ldots<q_{k-1}B_kq_k>$

$\Rightarrow^n_G c\, y \quad (= x)$.

Simulating PDAs by CFGs (cont'd)

(<=:) Suppose $\langle pB_1q_1\rangle\langle q_1 B_2 q_2\rangle\ldots\langle q_{k-1} B_k q\rangle \overset{L}{\rightarrow}{}^{n+1}_G x$.

Let $\langle pB_1q_1\rangle \rightarrow c \langle r_0 C_1 r_1\rangle \langle r_1 C_2 r_2\rangle \ldots \langle r_{m-1} C_m q_1\rangle \in P$ --(*

be the first rule applied. i.e., Then

$\langle p\ B_1\ q_1\rangle \langle q_1\ B_2\ q_2\rangle \ldots\langle q_{k-1}\ B_k q\rangle$

$\overset{L}{\rightarrow}_G c \langle r_0\ C_1\ r_1\rangle \langle r_1\ C_2\ r_2\rangle \ldots \langle r_{m-1}\ C_m\ q_1\rangle \langle q_1\ B_2\ q_2\rangle \ldots \langle q_{k-1}B$

$\overset{L}{\rightarrow}_G{}^n\ cy\ \ (= x\ )$

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

$(p,x,B_1\ldots B_k) \dashrightarrow_M (r_0, y, C_1C_2\ldots C_mB_2\ldots B_n)$ --- By (**)

$\dashrightarrow^n{}_M (q,e,e)$. -- ,by ind. hyp. QED

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

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

iff $(s,x,\perp) \dashrightarrow^*_M (t,e,e)$ ---- Lemma 25.1

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

- 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

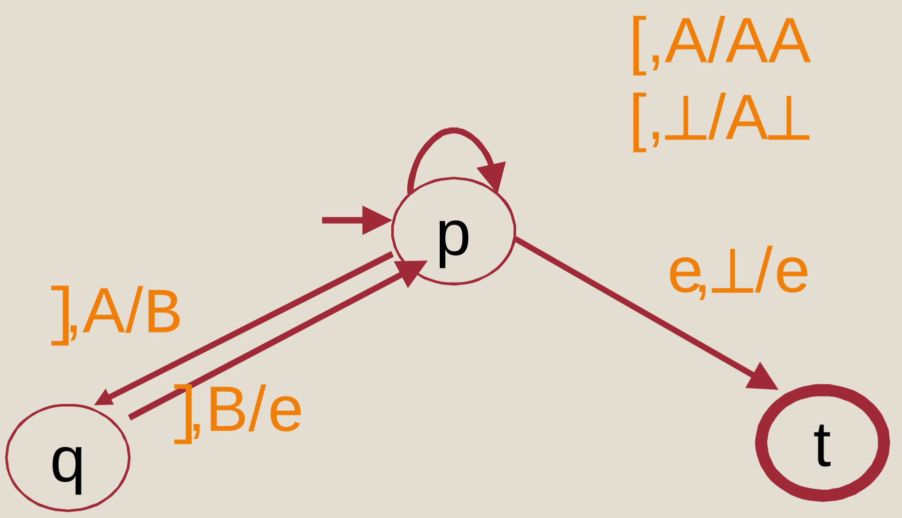
M = (Q, S, G, d, p, ⊥,{t}), where

Q = {p,q,t}, S = {[,]}, G = {A, B, ⊥},

and d is given as follows:

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

Example



- M can be simulated by the CFG G = (N,S, $<p\perp t>$, P) where
  - N = { $<X\ D\ Y>$ | X,Y $\in$ {p,q,t} and D $\in$ { A,B, $\perp$ } },
  - and P is derived from the following pseudo rules :
  - (p, [, $\perp$) --> (p, A$\perp$) : $<p \perp ?> \rightarrow [\ <pA\perp ?>$
  - (p,[,A) --> (p,AA) : $<p\ A\ ?_1> \rightarrow [\ <pA?_2A?_1>$
  - (p,], A) --> (q, B), : $<p\ A\ ?> \rightarrow]\ <qB?>$
  - This produce 3 rules (? = p or q or t).
  - (q,], B) --> (p, e), : $<q\ B\ ?> \rightarrow]\ <p\ e\ ?>$
  - This produces 1 rule :
  - (? = p, but could not be q or t why ?)
  - $<q\ B\ ?> \rightarrow]\ <p\ e\ ?> \Rightarrow <qBp> \rightarrow]\ <pep> \rightarrow^0]$
  - (p,e, $\perp$) --> (t,e) : $<p\perp ?> \rightarrow <t\ e\ ?>$
  - This results in $<p\perp t> \rightarrow e$ (since $<t\ e\ t> \rightarrow e$.)

- <p ⊥ ?> → [<pA⊥?> → resulting in 3 rules : ? = p, q or t.
- <p ⊥ p> → [<pA⊥p> ---(1)
- <p ⊥ q> → [<pA⊥q> ---(2)
- <p ⊥ t> → [<pA⊥t> ---(3)
- (1)~(3) each again need to be expanded into 3 rules.
- <pA⊥p> → <pA?><? ⊥ p> where ? is p or q or t.
- <pA⊥q> → <pA?><? ⊥ q> where ? is p or q or t.
- <pA⊥ t> → <pA?><? ⊥ t> where ? is p or q or t.
- <p A $?_1$> → [<pA$?_2$A$?_1$> resulting in 9 rules:
- Where $?_2$ = p,q, or t.
- <p A p> → [<pA$?_2$> <$?_2$⊥p> ---(1)
- <p A q> → [<pA$?_2$> <$?_2$⊥q> ---(2)
- <p A t> → [<pA$?_2$> <$?_2$⊥t> ---(3)