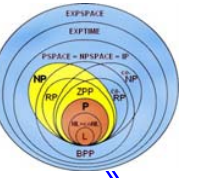
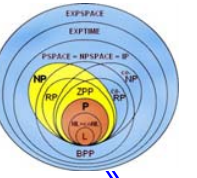


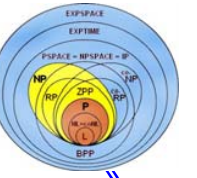
Session 8

- Ambiguity in Context-Free Grammars
- Pushdown Automata
- The Language of a Pushdown Automaton



Ambiguity

in Context-Free Grammars



Ambiguous Grammars

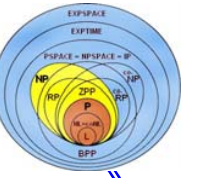
In the grammar

$$1.\, E \to I,\ 2.\, E \to E + E,\ 3.\, E \to E \times E,\ 4.\, E \to (E),$$

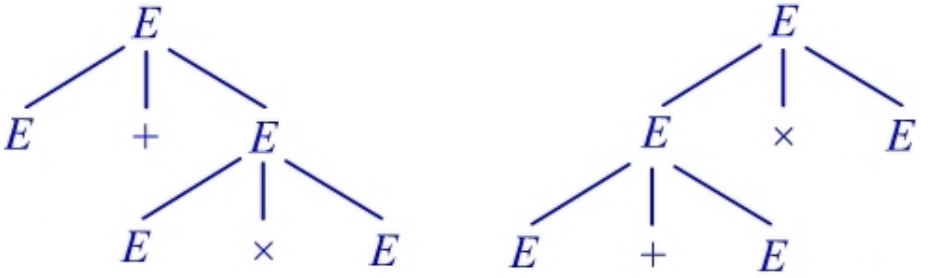
$$5.\, I \to a,\ 6.\, I \to b,\ 7.\, I \to Ia,\ 8.\, I \to Ib,\ 9.\, I \to I0,\ 10.\, I \to I1.$$

the sentential for $E + E \times E$ has two derivations:

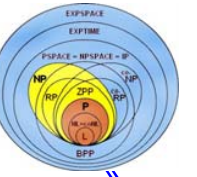
$$E \Rightarrow E + E \Rightarrow E + E \times E \quad \text{and} \quad E \Rightarrow E \times E \Rightarrow E + E \times E$$



This gives us two parse trees:



This grammar is not a good one for providing unique structure. To use this expression grammar in a compiler, we would have to modify it to provide only the correct groupings.



The mere existence of several derivations is not dangerous, it is the existence of several parse trees that ruins a grammar.

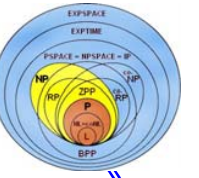
In the same grammar the string $a + b$ has several derivations, e.g.

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$$

and

$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$$

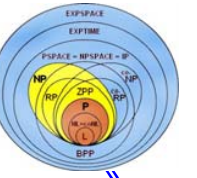
However, their parse trees are the same, and the structure of $a + b$ is unambiguous.



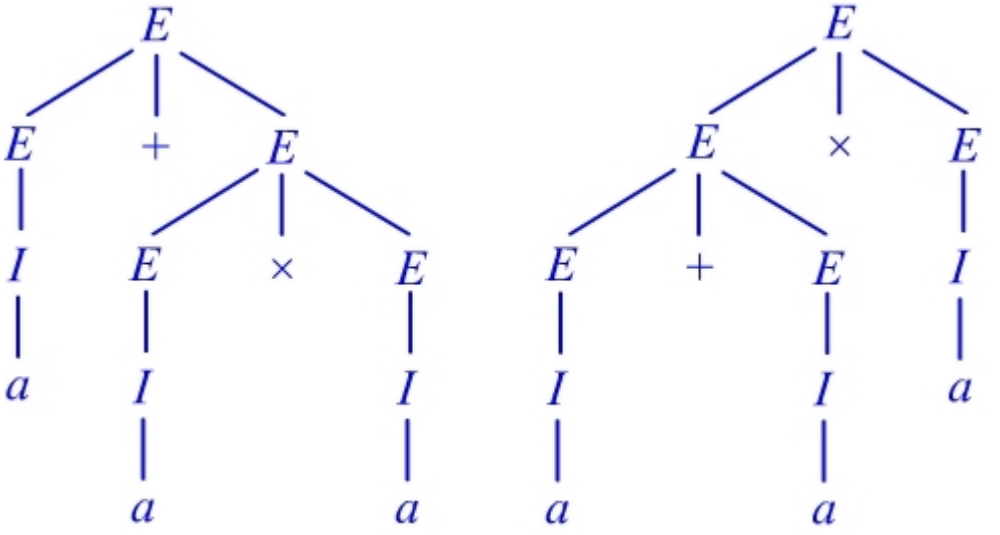
The two examples above suggested that it is not a multiplicity of derivations that cause ambiguity, but rather the existence of two or more parse trees.

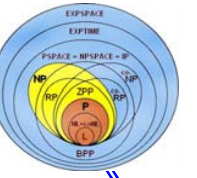
Let $G = (V, T, P, S)$ be a CFG. We say that G is ambiguous if there is at least one string w in T^* for which we can find two different parse trees, each with root labeled S and yield w.

If every string in $L(G)$ has at most one parse tree, G is said to be unambiguous.



Example In above grammar, the terminal string $a + a \times a$ has two parse trees:

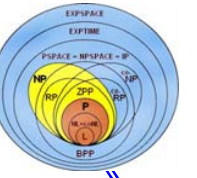




Removing Ambiguity from Grammars

- Good news: Sometimes we can remove ambiguity from CFG's "by hand".
- Bad news: There is no algorithm to do it.
- More bad news: There are context-free languages that have nothing but ambiguous CFG's; for these languages, removal of ambiguity is impossible.

Fortunately, in practice, for the sorts of constructs that appear in common programming languages, there are well-known techniques for eliminating ambiguity.

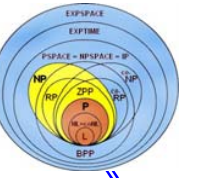


As an important illustration, we are studying the grammar

$$E \to I|E+E|E\times E|(E), \quad I \to a|b|Ia|Ib|I0|I1.$$

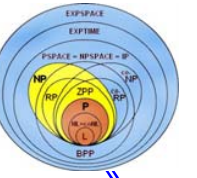
There are two causes of ambiguity:

1. There is no precedence between $\times$ and $+$.
2. There is no grouping of sequences of operators, e.g. is $E+E+E$ meant to be $E+(E+E)$ or $(E+E)+E$.



The solution to the problem of enforcing precedence is to introduce more variables, each representing expressions of same "binding strength".

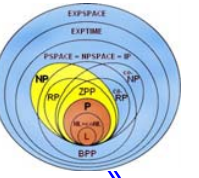
1. A *factor* is an expression that cannot be broken apart by an adjacent $\times$ or $+$. Our factors are identifiers and a parenthesized expression.
2. A *term* is an expression that cannot be broken by $+$. For instance $a \times b$ can be broken by $a1\times$ or $\times a1$, $a1 \times a \times b = (a1 \times a) \times b$. It cannot be broken by $+$, since e.g. $a1 + a \times b$ is (by precedence rules) same $a1 + (a \times b)$, and $a \times b + a1$ is same as $(a \times b) + a1$.
3. The rest are *expressions*, i.e. they can be broken apart with $\times$ or $+$.



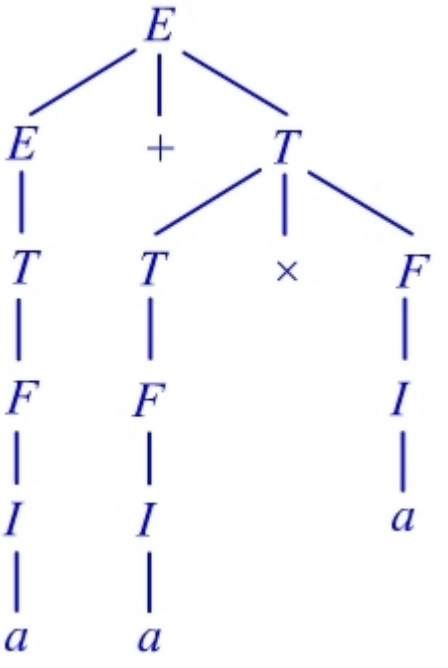
We'll let F stand for factors, T for terms, and E for expressions.

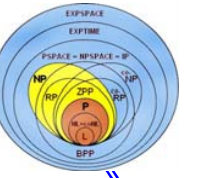
Consider the following grammar:

1. $I \rightarrow a|b|Ia|Ib|I0|I1$
2. $F \rightarrow I|(E)$
3. $T \rightarrow F|T \times F$
4. $E \rightarrow T|E + T$



Now the only parse tree for $a + a \times a$ will be

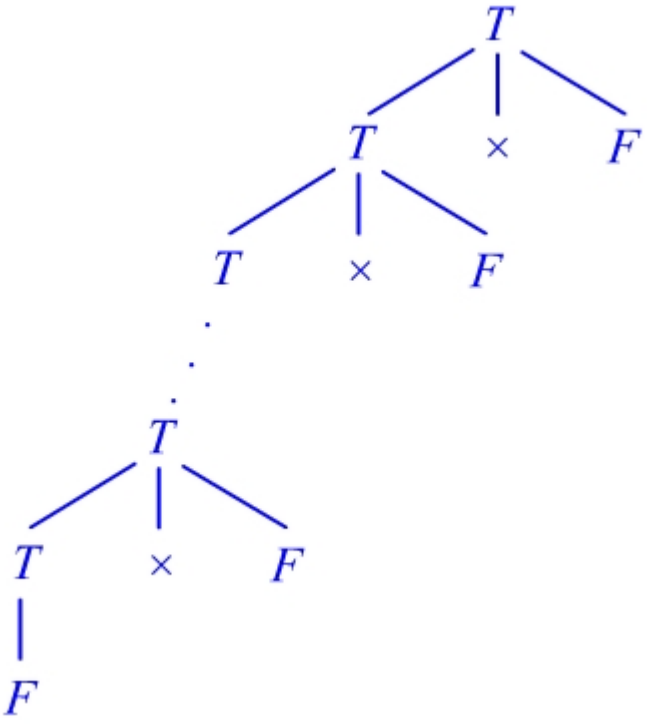


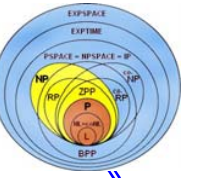


Why is the new grammar unambiguous?

Here is a intuitive explanation:

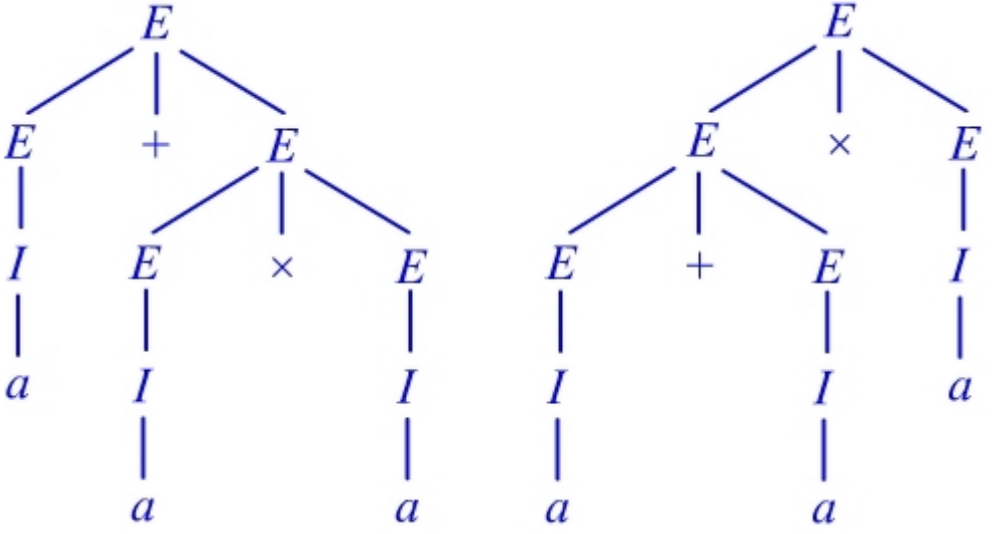
- A factor is either an identifier or (E), for some expression E.
- The only parse tree for a sequence $f_1 \times f_2 \times \cdots \times f_{n-1} \times f_n$ of factors is the one that gives $f_1 \times f_2 \times \cdots \times f_{n-1}$ as a term an f_n as a factor, as in the parse tree on the next slide.
- An expression is a sequence $t_1 + t_2 + \cdots + t_{n-1} + t_n$ of terms t_i. It can only be parsed with $t_1 + t_2 + \cdots + t_{n-1}$ as an expression and t_n as a term.

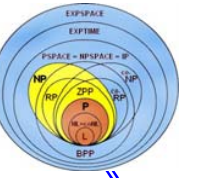

T
T × F
T × F
T
T × F
F



Leftmost Derivations and Ambiguity

Example The terminal string $a + a \times a$ has two parse trees:





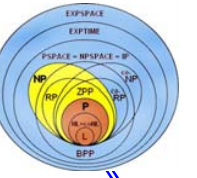
That give rise to two derivations

$$E \underset{lm}{\Rightarrow} E+E \underset{lm}{\Rightarrow} I+E \underset{lm}{\Rightarrow} a+E \underset{lm}{\Rightarrow} a+E\times E \underset{lm}{\Rightarrow} a+I\times E \underset{lm}{\Rightarrow} a+a\times E \underset{lm}{\Rightarrow} a+a\times I \underset{lm}{\Rightarrow} a+a\times a$$

and

$$E \underset{lm}{\Rightarrow} E\times E \underset{lm}{\Rightarrow} E+E\times E \underset{lm}{\Rightarrow} I+E\times E \underset{lm}{\Rightarrow} a+E\times E \underset{lm}{\Rightarrow} a+I\times E \underset{lm}{\Rightarrow} a+a\times E \underset{lm}{\Rightarrow}$$
$$a+a\times I \underset{lm}{\Rightarrow} a+a\times a$$

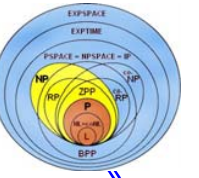
In general, there may be has many derivations for one parse tree. But many left-most/rightmost derivations implies many parse trees. That is, in an unambiguous grammar, leftmost/rightmost derivations will be unique.



Theorem 5.4 *For any CFG G, a terminal string w has two distinct parse trees if and only if w has two distinct leftmost derivation from the start symbol.*

Proof –SKETCH– (If). Let's look at how we construct a parse tree from a leftmost derivation. It should now be clear that two distinct derivations gives rise to two different parse trees.

(Only if). If the two parse trees differ, they have a node with different productions, say $A \to X_1X_2\cdots X_k$ and $A \to Y_1Y_2\cdots Y_m$. The corresponding leftmost derivations will use derivations based on these two different productions and will thus be distinct. ◀



Inherent Ambiguity

A CFL L is inherent ambiguous if *all* grammars for L are ambiguous.

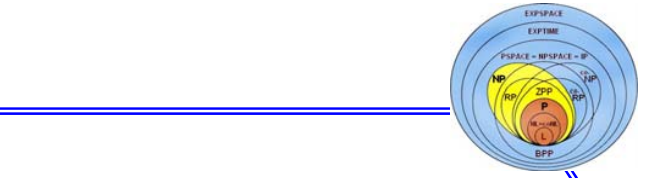
Example Consider $L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$.

A grammar for L is

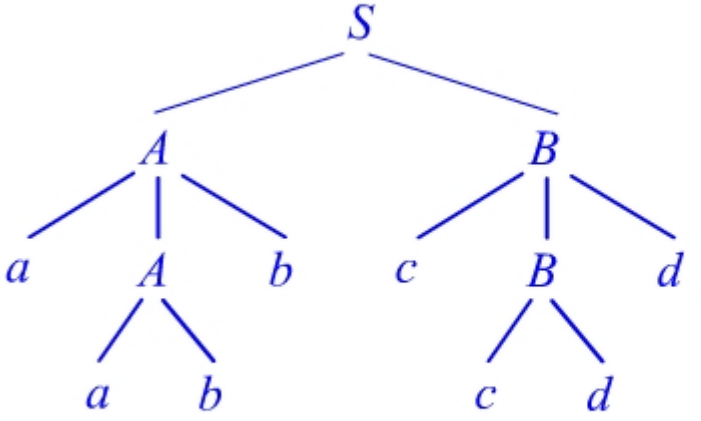
$$G = (\{A, B, C, D, S\}, \{a, b, c, d\}, P, S)$$

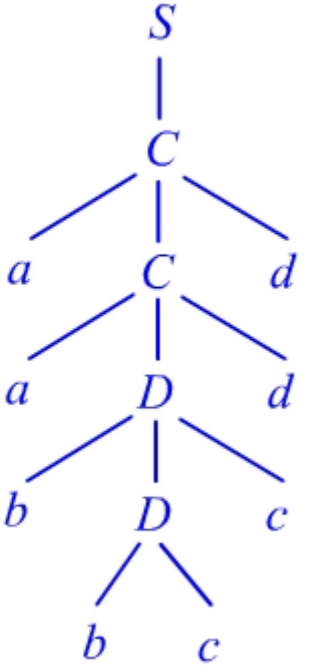
where P is as follows:

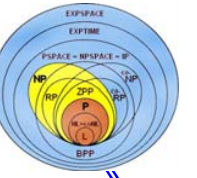
$$S \to AB|C,\ A \to aAb|ab,\ B \to cBd|cd,\ C \to aCd|aDd,\ D \to bDc|bc$$



Let's look at parsing the string *aabbccdd*







From this we see that there are two leftmost derivations:

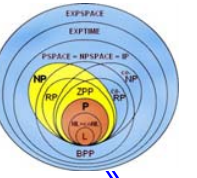
$$S \underset{lm}{\Rightarrow} AB \underset{lm}{\Rightarrow} aAbB \underset{lm}{\Rightarrow} aabbB \underset{lm}{\Rightarrow} aabbcBd \underset{lm}{\Rightarrow} aabbccdd$$

and

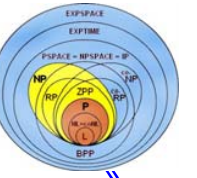
$$S \underset{lm}{\Rightarrow} C \underset{lm}{\Rightarrow} aCd \underset{lm}{\Rightarrow} aaDdd \underset{lm}{\Rightarrow} aabDcdd \underset{lm}{\Rightarrow} aabbccdd$$

It can be shown that *every* grammar for L behaves like the one above. The proof is complex.

The language L is inherently ambiguous.

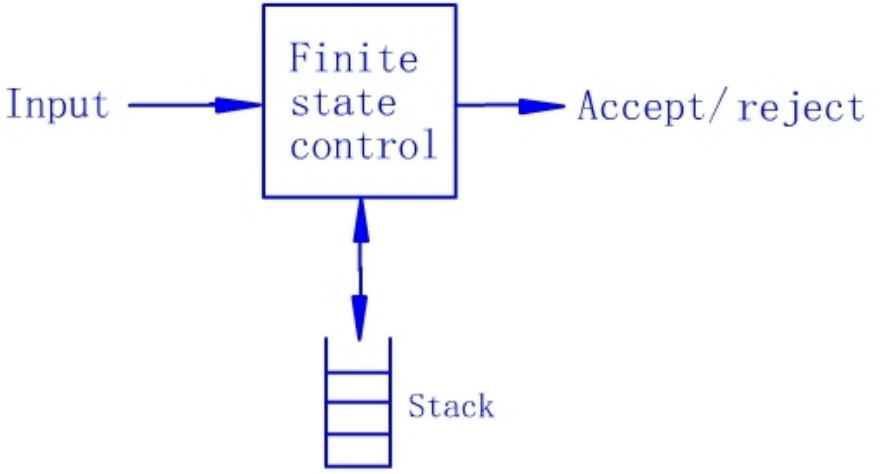


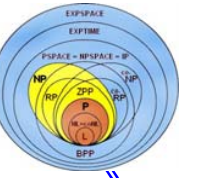
Pushdown Automata



Definition of PDA

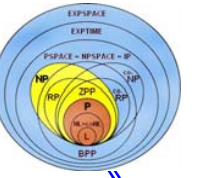
A pushdown automata (PDA) is essentially an ϵ-NFA with a stack.





On a transition the PDA:

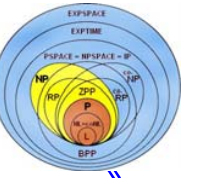
1. Consumes an input symbol.
2. Goes to a new state (or stays in the old).
3. Replaces the symbol at the top of the stack by any string. (It could be ϵ, pops the stack; same symbol, dose nothing; one other symbol, changes the top of the stack but does not push or pop it; two or more symbols, changes the top stack symbol, and then pushes one or more new symbols onto the stack.)



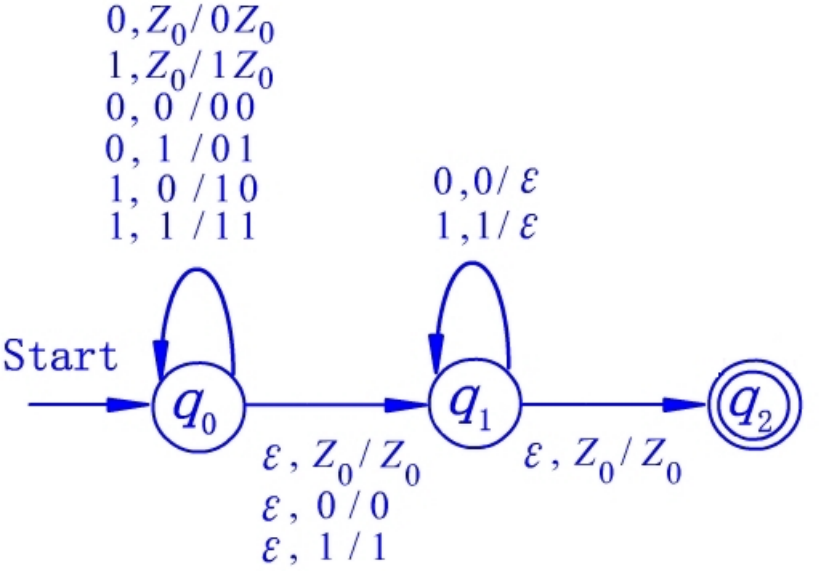
Example Let's consider $L_{wwr} = \{ww^R \mid w \in \{0,1\}^*\}$ with grammar $S \rightarrow 0S0|1S1|\epsilon$.

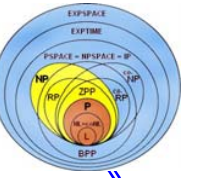
A PDA for L_{wwr} has three states, and operates as follows:

1. Guess that you're reading w. Stay in q_0, and push the input symbol onto the stack.
2. Guess that you're in the middle of ww^R. Go spontaneously to state q_1.
3. You're reading the head of w^R. Compare it to the top of the stack. If match, pop the stack, and remain in state q_1. If non't match, go to sleep.
4. If the stack is empty, go to state q_2 and accept.



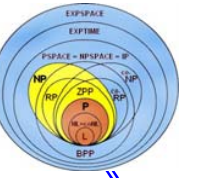
The PDA for L_{wwr} as a transition diagram:



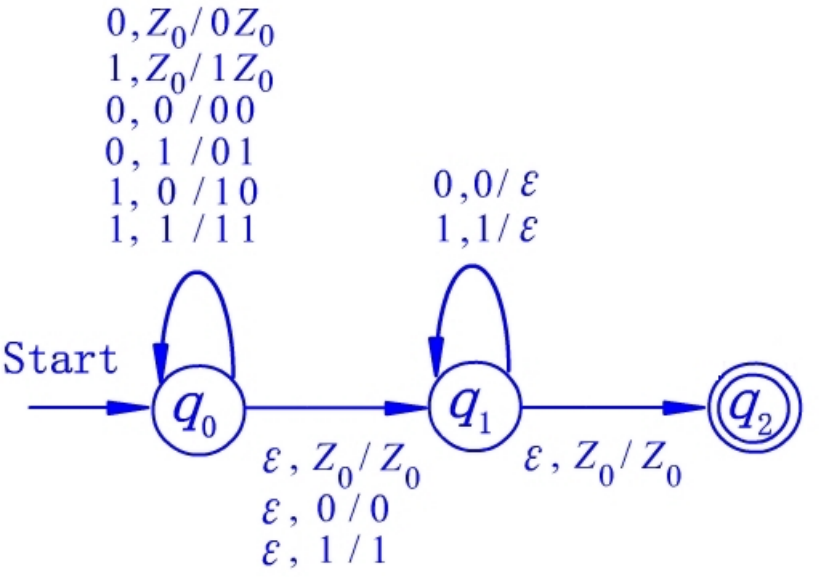


A Pushdown Automaton (PDA) is a 7-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

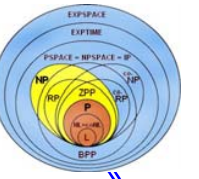
- Q is a finite set of *states*.
- Σ is a finite *input alphabet*.
- Γ is a finite *stack alphabet*.
- δ is a *transition function* from $Q \times \Sigma \cup \{\epsilon\} \times \Gamma$ to $2^{Q \times \Gamma^*}$.
- q_0 is a *start state*, a member of Q.
- Z_0 is the *start symbol* for the stack bing in Γ.
- F is a set of *accepting* states, a subset of Q.



Example The PDA



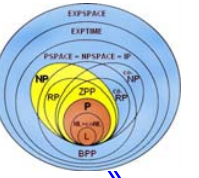
is actually the seven-tuple as follows.



$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is given by the following table (set brackets missing):

	$0, Z_0$	$1, Z_0$	$0, 0$	$0, 1$	$1, 0$	$1, 1$	ϵ, Z_0	$\epsilon, 0$	$\epsilon, 1$
$\rightarrow q_0$	$q_0, 0Z_0$	$q_0, 1Z_0$	$q_0, 00$	$q_0, 01$	$q_0, 10$	$q_0, 11$	q_1, Z_0	$q_1, 0$	$q_1, 1$
q_1	–	–	q_1, ϵ	–	–	q_1, ϵ	q_2, Z_0	–	–
$\star q_2$	–	–	–	–	–	–	–	–	–



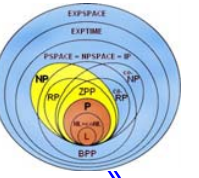
Instantaneous Description

A PDA goes from configuration to configuration when consuming input.

To reason about PDA computation, we use instantaneous description (ID) of the PDA. An ID is a triple

$$(q, w, \gamma)$$

where q is the state, w the remaining input, and γ the stack contents.



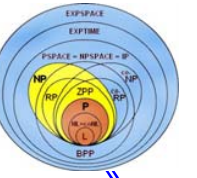
Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $\forall\, w \in \Sigma^*, \beta \in \Gamma^*$:

$$(p, \alpha) \in \delta(q, a, X) \Rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta).$$

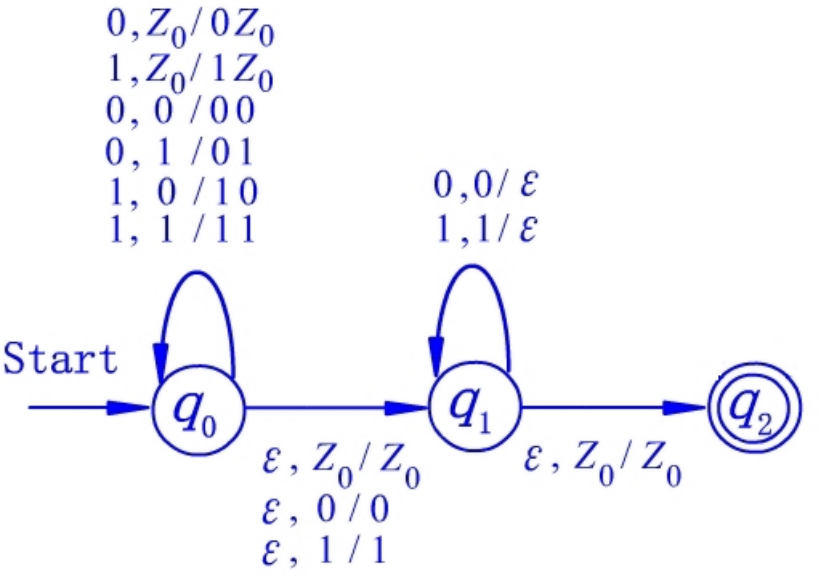
We define $\overset{*}{\vdash}$ to be the reflexive-transitive closure of $\vdash$:

Basis step: $I \overset{*}{\vdash} I$ for any ID I.

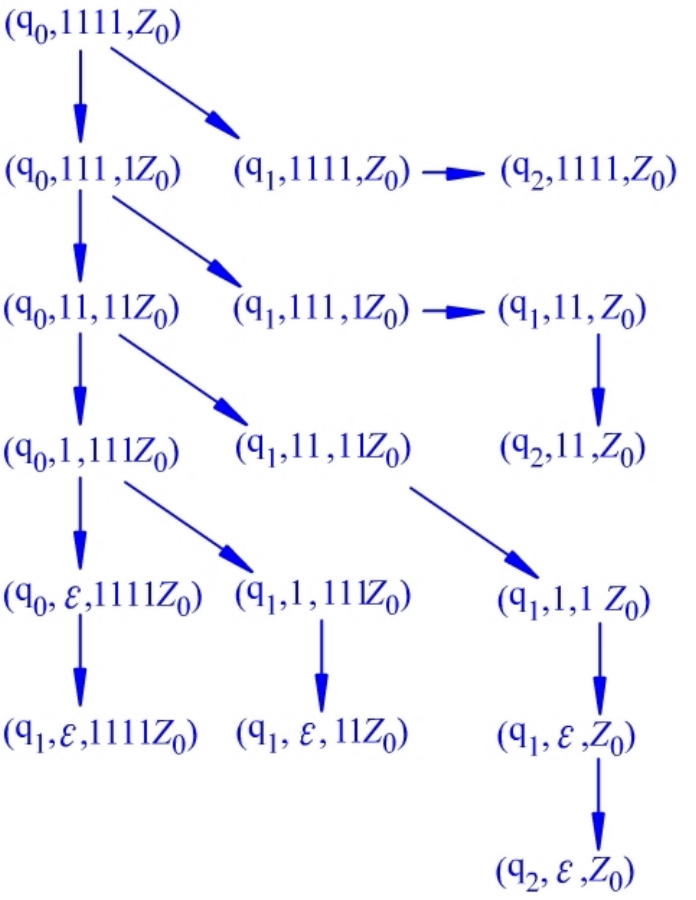
Inductive step: $I \overset{*}{\vdash} J$ if there exists some ID K such that $I \vdash K$ and $K \overset{*}{\vdash} J$.

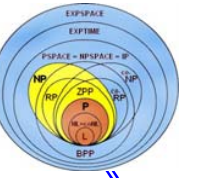


Example On input 1111 the PDA



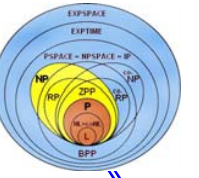
has the following computation sequences.


$(q_0,1111,Z_0)$
$(q_0,111,1Z_0)$
$(q_1,1111,Z_0)$
$(q_2,1111,Z_0)$
$(q_0,11,11Z_0)$
$(q_1,111,1Z_0)$
$(q_1,11,Z_0)$
$(q_0,1,111Z_0)$
$(q_1,11,11Z_0)$
$(q_2,11,Z_0)$
$(q_0,\varepsilon,1111Z_0)$
$(q_1,1,111Z_0)$
$(q_1,1,1\ Z_0)$
$(q_1,\varepsilon,1111Z_0)$
$(q_1,\varepsilon,11Z_0)$
(q_1,ε,Z_0)
(q_2,ε,Z_0)



There are three important principles about ID's and their transitions:

1. If an ID sequence is a legal computation for a PDA, then so is the sequence obtained by adding an additional string at the end of component number two.
2. If an ID sequence is a legal computation for a PDA, then so is the sequence obtained by adding an additional string at the bottom of component number three.
3. If an ID sequence is a legal computation for a PDA, and some tail of the input is not consumed, then removing this tail from all ID's result in a legal computation sequence.



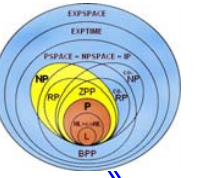
Theorem 6.1 *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $\forall\, w \in \Sigma^*, \gamma \in \Gamma^*$:*

$$(q, x, \alpha) \overset{*}{\vdash} (p, y, \beta) \Rightarrow (q, xw, \alpha\gamma) \overset{*}{\vdash} (p, yw, \beta\gamma).$$

Proof Introduction on the number of steps in the sequence of ID's that take (q, x, α) to (p, y, β). ◄

Note that if $\gamma = \epsilon$ we have the first principle above, and if $w = \epsilon$, then we have the second principle.

☞ The reverse of the theorem is false.



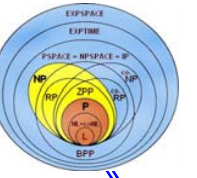
We state the third principle formally as:

Theorem 6.2 *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA.*

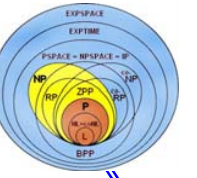
$$(q, xw, \alpha) \overset{*}{\vdash} (p, yw, \beta) \Rightarrow (q, x, \alpha) \overset{*}{\vdash} (p, y, \beta).$$

☞ Although a FA has no stack, we could use a pair (q, w) as the ID of a finite automaton. For any FA, we could show that $\hat{\delta}(q, w) = p$ if and only if $(q, wx) \overset{*}{\vdash} (p, x)$ for all strings x. The fact that x can be anything we wish without influencing the behavior of the FA is a theorem analogous to Theorem 6.1 and 6.2.


Break for 15 minutes

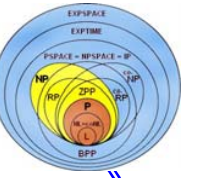


The Language of a Pushdown Automaton



We have two approaches to defining the language of a PDA: "acceptance by final state" and "acceptance by empty stack".

- These two methods are equivalent, in the sense that a language L has a PDA that accepts it by final state iff L has a PDA that accepts it by empty stack.
- For given PDA P, the languages that P accepts by final state and by empty stack are usually different.



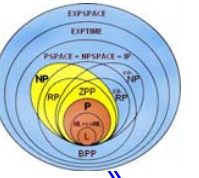
Acceptance by the Final State

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language accepted by P by final state is

$$L(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \alpha), q \in F\}.$$

Example Below PDA accepts exactly L_{wwr} by final state.

	$0, Z_0$	$1, Z_0$	$0, 0$	$0, 1$	$1, 0$	$1, 1$	ϵ, Z_0	$\epsilon, 0$	$\epsilon, 1$
$\rightarrow q_0$	$q_0, 0Z_0$	$q_0, 1Z_0$	$q_0, 00$	$q_0, 01$	$q_0, 10$	$q_0, 11$	q_1, Z_0	$q_1, 0$	$q_1, 1$
q_1	–	–	q_1, ϵ	–	–	q_1, ϵ	q_2, Z_0	–	–
$\star q_2$	–	–	–	–	–	–	–	–	–

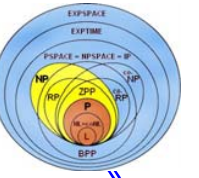


Let $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$ be the machine. We prove that $L(P) = L_{wwr}$.

($\supseteq$-direction) Let $x \in L_{wwr}$. Then $x = ww^R$, and the following is a legal computation sequence

$$(q_0, ww^R, Z_0) \overset{*}{\vdash} (q_0, w^R, w^R Z_0) \vdash (q_1, w^R, w^R Z_0) \overset{*}{\vdash} (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0).$$

($\subseteq$-direction) Observe that the only way the PDA can enter q_2 is if it is in state q_1 with an empty stack. Thus it is sufficient to show that if $(q_0, x, Z_0) \overset{*}{\vdash} (q_1, \epsilon, Z_0)$ then $x = ww^R$, for some word w.



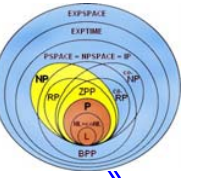
We'll show by induction on $|x|$ that

$$(q_0, x, \alpha) \overset{*}{\vdash} (q_1, \epsilon, \alpha) \quad \Rightarrow \quad x = ww^R.$$

Basis step: If $x = \epsilon$ then x is a palindrome.

Inductive step: Suppose $x = a_1 a_2 \cdots a_n$, where $n > 0$, and the induction hypothesis holds for shorter strings. There are two moves for the PDA from ID(q_0, x, α):

Move 1: The spontaneous $(q_0, x, \alpha) \vdash (q_1, x, \alpha)$. Now $(q_1, x, \alpha) \overset{*}{\vdash} (q_1, \epsilon, \beta)$ implies that $|\beta| < |\alpha|$, which implies $\beta \neq \alpha$.



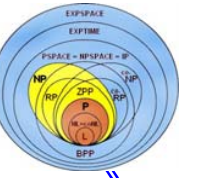
Move 2: Loop and push $(q_0, a_1a_2\cdots a_n, \alpha) \vdash (q_0, a_2\cdots a_n, a_1\alpha)$. In this case there is a sequence

$$(q_0, a_1a_2\cdots a_n, \alpha) \vdash (q_0, a_2\cdots a_n, a_1\alpha) \vdash \cdots \vdash (q_1, a_n, a_1\alpha) \vdash (q_1, \epsilon, \alpha).$$

Thus $a_1 = a_n$ and $(q_0, a_2\cdots a_n, a_1\alpha) \overset{*}{\vdash} (q_1, a_n, a_1\alpha)$. By Theorem 6.2 we can remove a_n. Therefore

$$(q_0, a_2\cdots a_{n-1}, a_1\alpha) \overset{*}{\vdash} (q_1, \epsilon, a_1\alpha).$$

Then, by the induction hypothesis $a_2\cdots a_{n-1} = yy^R$. Then $x = a_1yy^Ra_n$ is a palindrome. ◀



Acceptance by the Empty Stack

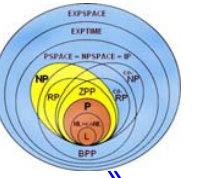
Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language accepted by P by empty stack is

$$N(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)\}$$

for any state q.

Question How to modify the palindrome PDA to accept by empty stack?

Since the set of accepting states is irrelevant, we would write a PDA P as a six-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ if all we care about is the language that P accepts by empty stack.



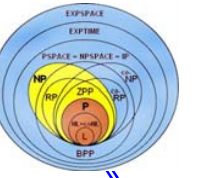
From Empty Stack to Final State

Theorem 6.3 *If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there exists a PDA P_F such that $L = L(P_F)$.*

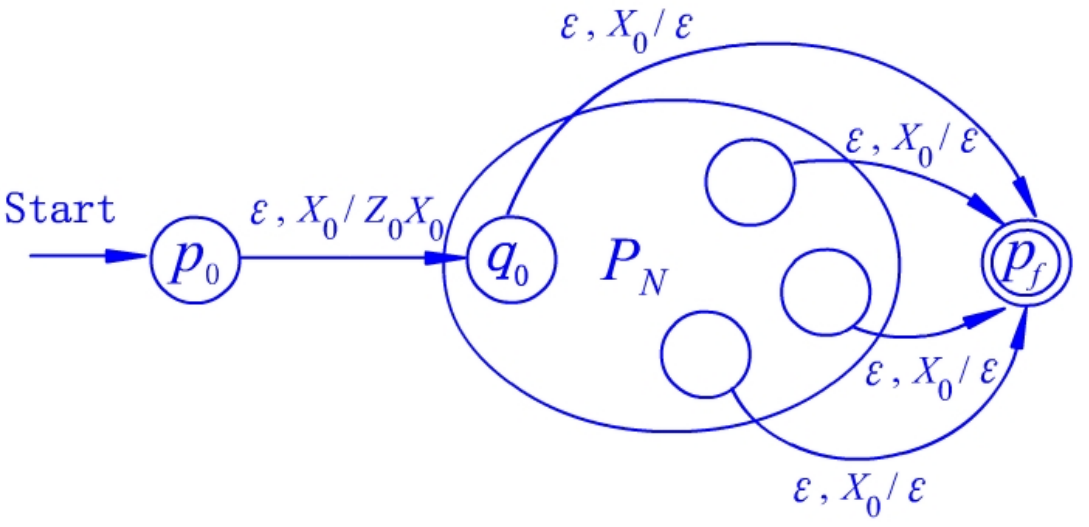
Proof Let $P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$ where $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$, and for all $q \in Q, a \in \Sigma \cup \{\epsilon\}, Y \in \Gamma$

$$\delta_F(q, a, Y) = \delta_N(q, a, Y),$$

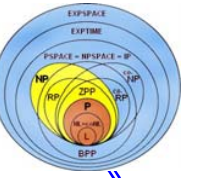
and in addition $(p_f, \epsilon) \in \delta_F(q, \epsilon, X_0)$.



The diagram is



We have show that $L(P_F) = N(P_N)$.



($\subseteq$-direction) By inspecting the diagram.

($\supseteq$-direction) Let $w \in N(P_N)$. Then $(q_0, w, Z_0) \overset{*}{\underset{N}{\vdash}} (q, \epsilon, \epsilon)$, for some q. From Theorem 6.1 we get

$$(q_0, w, Z_0X_0) \overset{*}{\underset{N}{\vdash}} (q, \epsilon, X_0).$$

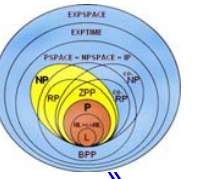
Since $\delta_N \subset \delta_F$ we have

$$(q_0, w, Z_0X_0) \overset{*}{\underset{F}{\vdash}} (q, \epsilon, X_0).$$

We conclude that

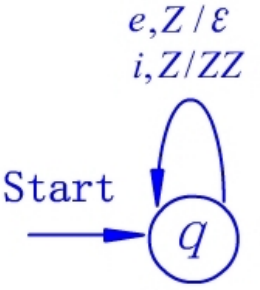
$$(p_0, w, X_0) \underset{F}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{F}{\vdash}} (q, \epsilon, X_0) \underset{F}{\vdash} (p_f, \epsilon, \epsilon).$$

◀



Example Let's design P_N for catching errors in strings meant to be in the *if-else*-grammar G: $S \to \epsilon | SS | iS | iSe$.

The diagram for P_N is



Formally, $P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z)$, where

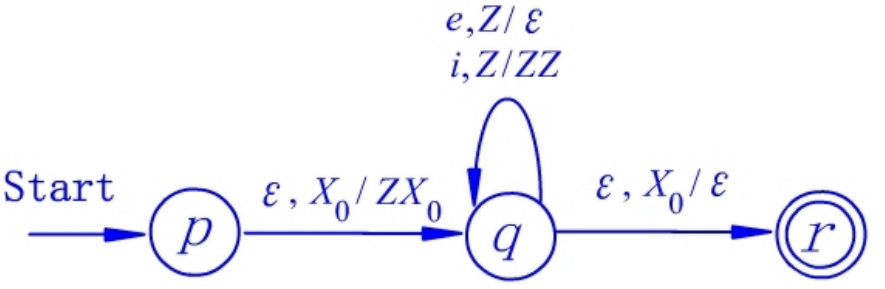
$$\delta_N(q, i, Z) = \{(q, ZZ)\} \quad \delta_N(q, e, Z) = \{(q, \epsilon)\}.$$

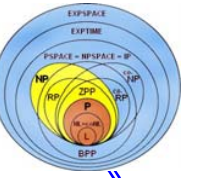
From P_N we can construct $P_F = (\{p, q, r\}, \{i, e\}, \{Z, X_0\}, \delta_F, p, X_0, \{r\})$, where

$$\delta_F(p, \epsilon, X_0) = \{(q, ZX_0)\},\ \ \delta_F(q, i, Z) = \delta_N(q, i, Z) = \{(q, ZZ)\},$$

$$\delta_F(q, e, Z) = \delta_N(q, e, Z) = \{(q, \epsilon)\},\ \ \delta_F(q, \epsilon, X_0) = \{(r, \epsilon)\}$$

The diagram for P_F is





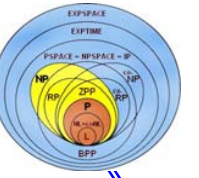
From Final State to Empty Stack

Theorem 6.4 *Let $L = L(P_F)$, for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.*

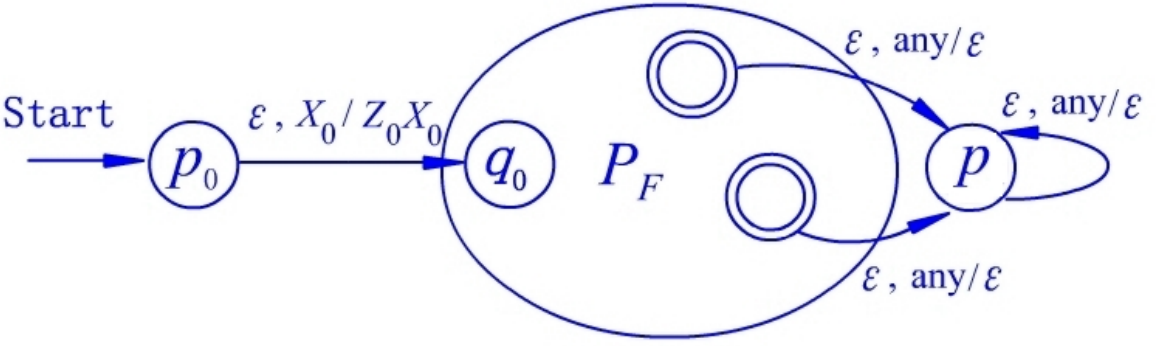
Proof Let $P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$, where $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$ for $Y \in \Gamma \cup \{X_0\}$, and for all $q \in Q, a \in \Sigma \cup \{\epsilon\}, Y \in \Gamma$

$$\delta_N(q, a, Y) = \delta_F(q, a, Y),$$

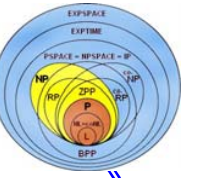
and in addition $\forall\, q \in F$, and $Y \in \Gamma \cup \{X_0\}$: $(p, \epsilon) \in \delta_N(q, \epsilon, Y)$.



The diagram is



We have show that $N(P_N) = L(P_F)$.



($\subseteq$-direction) By inspecting the diagram.

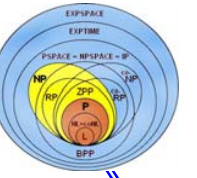
($\supseteq$-direction) Let $w \in L(P_F)$. Then $(q_0, w, Z_0) \overset{*}{\underset{F}{\vdash}} (q, \epsilon, \alpha)$, for some $q \in F$, $\alpha \in \Gamma^*$. Since $\delta_F \subset \delta_N$, and Theorem 6.1 says that X_0 can be slid under the stack, we get

$$(q_0, w, Z_0X_0) \overset{*}{\underset{N}{\vdash}} (q, \epsilon, \alpha X_0).$$

The P_N can be compute

$$(p_0, w, X_0) \underset{N}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{N}{\vdash}} (q, \epsilon, \alpha X_0) \underset{N}{\vdash} (p, \epsilon, \epsilon).$$

◀



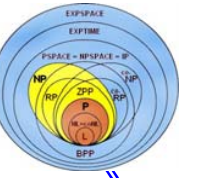
Example What languages is accepted by following PDA with final state?

$$P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

with transitions $\delta(q_0, a, Z_0) = \{(q_1, a), (q_2, \epsilon)\}$, $\delta(q_1, b, a) = \{(q_1, b)\}$, $\delta(q_1, b, b) = \{(q_1, b)\}$, $\delta(q_1, a, b) = \{(q_2, \epsilon)\}$.

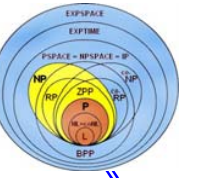
Solution $L = \{a\} \cup L(abb^*a)$.

What language accepted by empty stack?


EXPSPACE
EXPTIME
PSPACE = NPSPACE = IP
NP
RP
ZPP
P
NP
RP
L
BPP

Thank you!


華東師範大學
EAST CHINA NORMAL UNIVERSITY


EXPSPACE
EXPTIME
PSPACE = NPSPACE = IP
NP
RP
ZPP
P
L
BPP


華東師範大學
EAST CHINA NORMAL UNIVERSITY