Chapter 6

Pushdown Automata

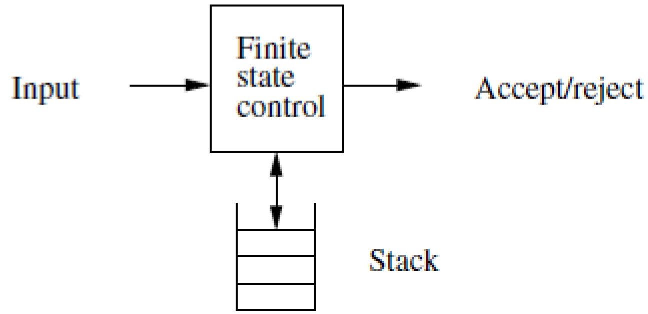
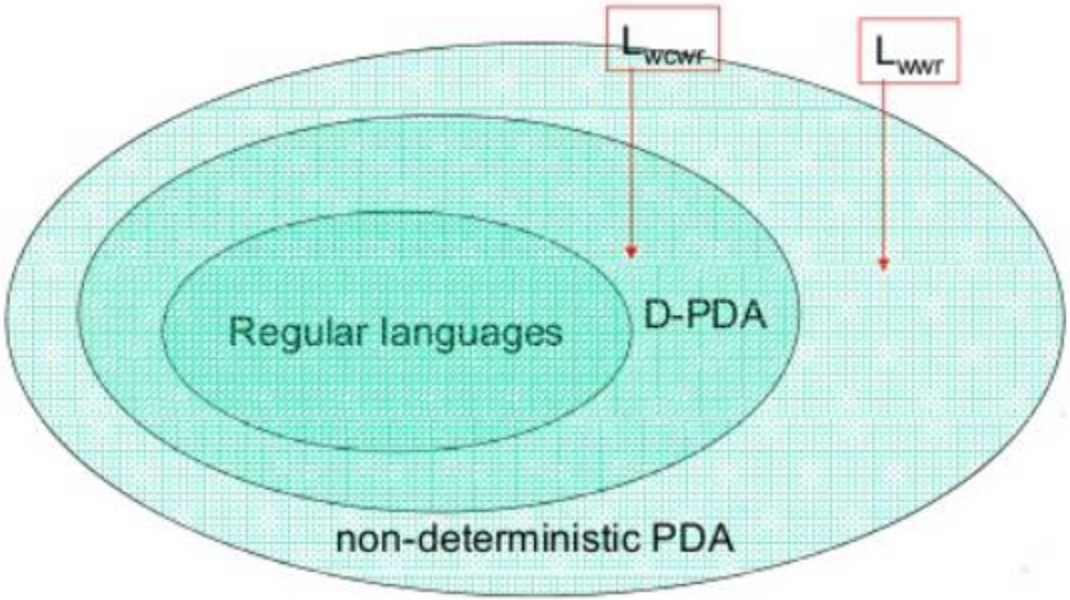


Figure 6.1: A pushdown automaton is essentially a finite automaton with a stack data structure

- PDA is an extension of nondeterministic finite automaton with a stack (of infinite size).
- DPDA -- deterministic version of PDA is not enough to recognize CFLs.



It holds that:

$$\left\{\begin{array}{c}\text{Deterministic}\\ \text{Context-Free}\\ \text{Languages}\\ \text{(DPDA)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{PDAs}\end{array}\right\}$$

Since every DPDA is also a PDA

6.1.2 The Formal Definition of Pushdown Automata

Our formal notation for a *pushdown automaton* (PDA) involves seven components. We write the specification of a PDA P as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:

Q: A finite set of *states*, like the states of a finite automaton.

Σ: A finite set of *input symbols*, also analogous to the corresponding component of a finite automaton.

Γ: A finite *stack alphabet*. This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

δ: The *transition function*.

q_0: The *start state*. The PDA is in this state before making any transitions.

Z_0: The *start symbol*. Initially, the PDA's stack consists of one instance of this symbol, and nothing else.

F: The set of *accepting states*, or *final states*.

δ: The *transition function*. As for a finite automaton, δ governs the behavior of the automaton. Formally, δ takes as argument a triple $\delta(q, a, X)$, where:

1. q is a state in Q.
2. a is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol.
3. X is a stack symbol, that is, a member of Γ.

The output of δ is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols that replaces X at the top of the stack. For instance, if $\gamma = \epsilon$, then the stack is popped, if $\gamma = X$, then the stack is unchanged, and if $\gamma = YZ$, then X is replaced by Z, and Y is pushed onto the stack.

- $\delta(q, a, X) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots\}$ where $\gamma_i \in \Gamma^*$.
  - Note, it is possible that $\gamma_i = \epsilon$.
  - $p_i \in Q$.
- $a \in \Sigma \cup \{\epsilon\}$.
- $X \in \Gamma$
  - X can never be ϵ. PDA must read a symbol from stack.

$(p, \boxed{\gamma})$

If $\gamma = YZ$, then Y will be on the top of the stack.
If $\gamma = \epsilon$, then do not push anything on to the stack.

L_{wwr}

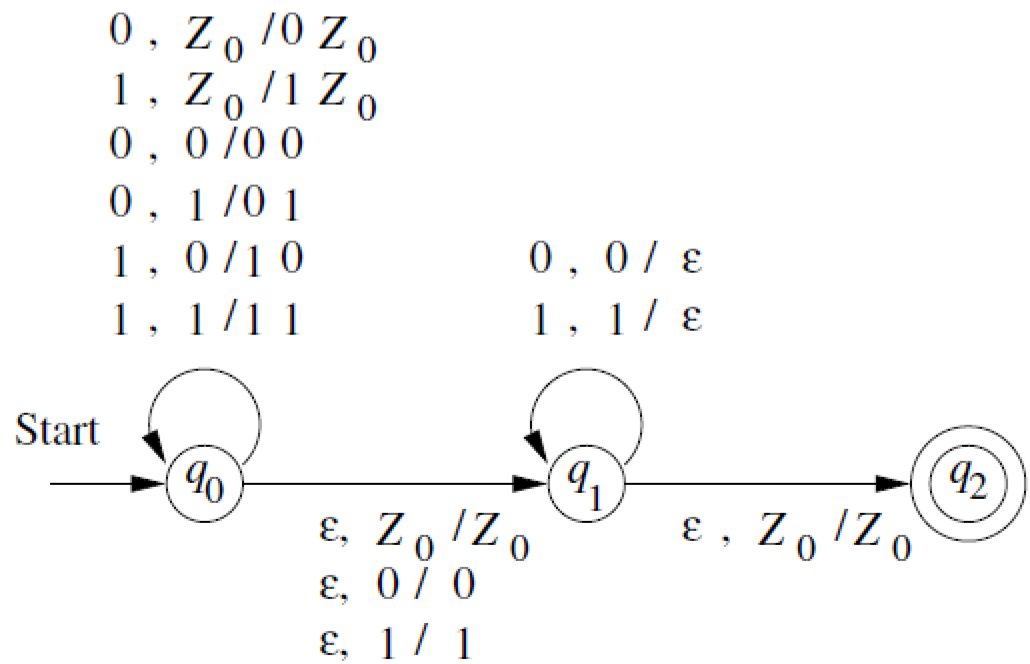


Figure 6.2: Representing a PDA as a generalized transition diagram

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

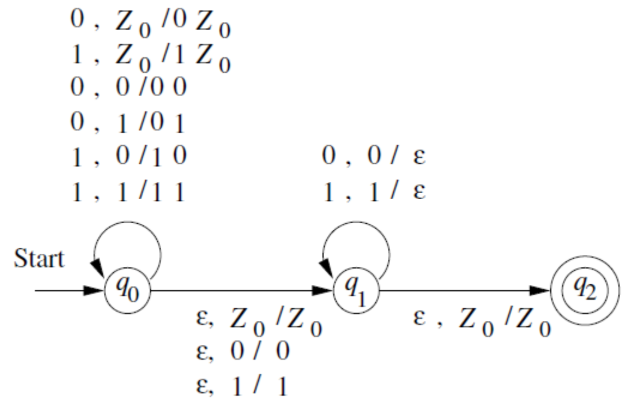


Figure 6.2: Representing a PDA as a generalized transition diagram

1. $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$. One of these rules applies initially, when we are in state q_0 and we see the start symbol Z_0 at the top of the stack. We read the first input, and push it onto the stack, leaving Z_0 below to mark the bottom.

2. $\delta(q_0, 0, 0) = \{(q_0, 00)\}$, $\delta(q_0, 0, 1) = \{(q_0, 01)\}$, $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, and $\delta(q_0, 1, 1) = \{(q_0, 11)\}$. These four, similar rules allow us to stay in state q_0 and read inputs, pushing each onto the top of the stack and leaving the previous top stack symbol alone.

3. $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$, $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$. These three rules allow P to go from state q_0 to state q_1 spontaneously (on ϵ input), leaving intact whatever symbol is at the top of the stack.

4. $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$, and $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$. Now, in state q_1 we can match input symbols against the top symbols on the stack, and pop when the symbols match.

5. $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$. Finally, if we expose the bottom-of-stack marker Z_0 and we are in state q_1, then we have found an input of the form ww^R. We go to state q_2 and accept.

6.1.4 Instantaneous Descriptions of a PDA

we shall represent the configuration of a PDA by a triple (q, w, γ), where

1. q is the state,
2. w is the remaining input, and
3. γ is the stack contents.

One step, and several steps

Suppose $\delta(q, a, X)$ contains (p, α).

Then for all strings w in Σ^* and β in Γ^*: $\quad (q, aw, X\beta) \vdash (p, w, \alpha\beta)$

$\boxed{\overset{*}{\vdash} \text{ is reflexive and transitive closure of } \vdash}$

6.2 The Languages of a PDA

6.2.1 Acceptance by Final State

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

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

for some state q in F and any stack string α.

6.2 The Languages of a PDA

6.2.1 Acceptance by Final State

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

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

for some state q in F and any stack string α.

That is, starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

6.2 The Languages of a PDA

6.2.1 Acceptance by Final State

Do we have some other acceptance criterion?!

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

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

for some state q in F and any stack string α.

That is, starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

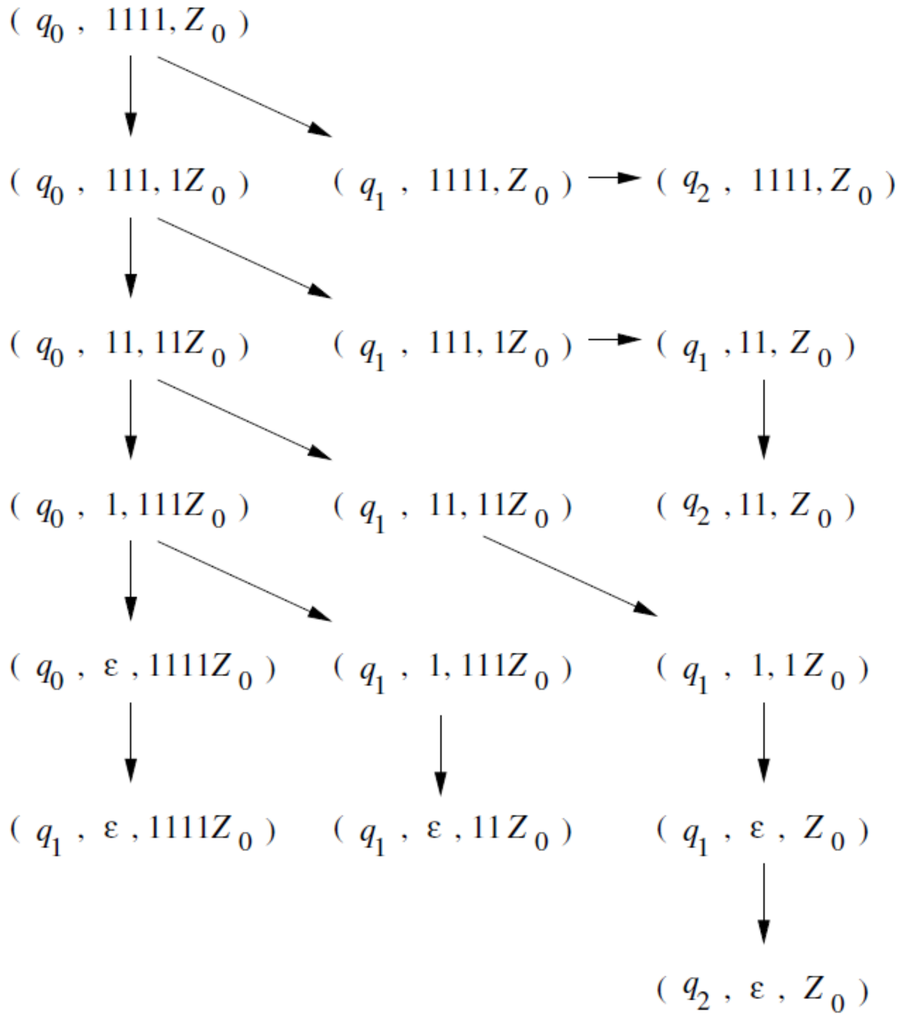
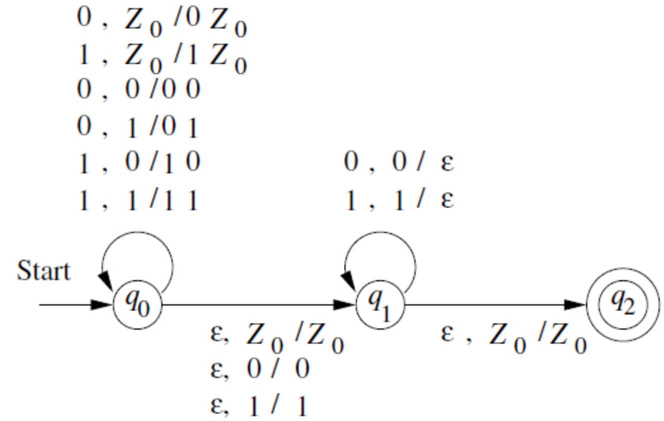


Figure 6.3: ID's of the PDA of Example 6.2 on input 1111



Representing a PDA as a generalized transition diagram

$$(q_0, 1111, Z_0) \vdash (q_0, 111, 1Z_0) \vdash (q_0, 11, 11Z_0) \vdash (q_1, 11, 11Z_0) \vdash (q_1, 1, 1Z_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0).$$

$$\boxed{\begin{array}{l}(q_0, 1111, Z_0) \vdash (q_0, 111, 1Z_0) \vdash (q_0, 11, 11Z_0) \vdash (q_1, 11, 11Z_0) \vdash \\ (q_1, 1, 1Z_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0).\end{array}}$$

- This is same as

$$(q_0, 1111, Z_0) \overset{*}{\vdash} (q_2, \epsilon, Z_0).$$

- So, 1111 is in the language.

6.2 The Languages of a PDA

6.2.1 Acceptance by Final State

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

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

for some state q in F and any stack string α.

6.2.2 Acceptance by Empty Stack

For each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, we also define

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

for any state q. That is, $N(P)$ is the set of inputs w that P can consume and at the same time empty its stack.[2]

$L(P)$ Vs. $N(P)$

- For the same PDA P there are now two languages !!

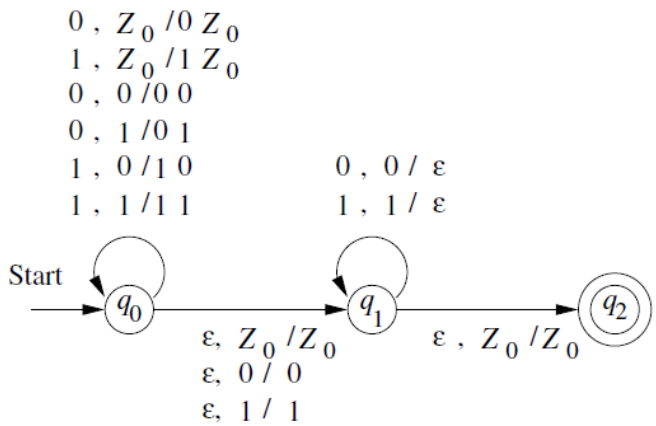


Figure 6.2: Representing a PDA as a generalized transition diagram

- $L(P) = \{ww^R | w \in (0+1)^*\} = L_{wwr}$
- $N(P) = ?$

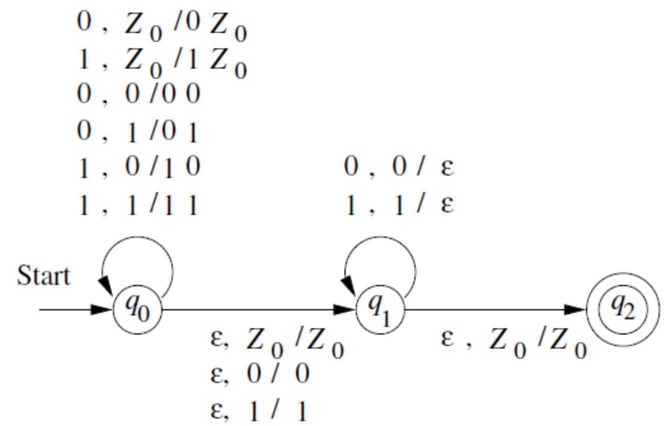


Figure 6.2: Representing a PDA as a generalized transition diagram

- $L(P) = \{ww^R | w \in (0+1)^*\} = L_{wwr}$
- $N(P) = \phi$
  - Stack never becomes empty
  - But, a small change can make $N(P) = L(P)$.
  - What is that?

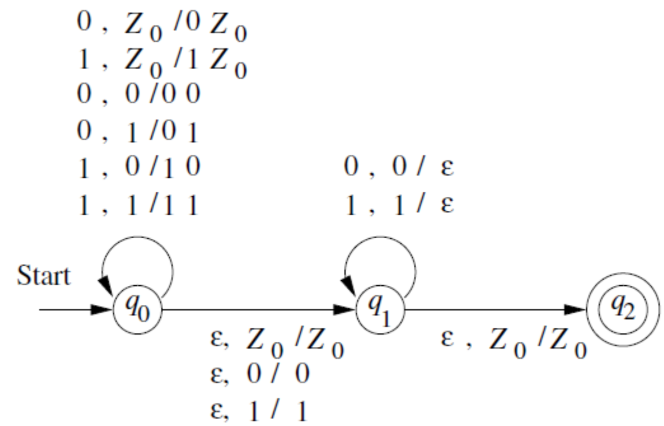


Figure 6.2: Representing a PDA as a generalized transition diagram

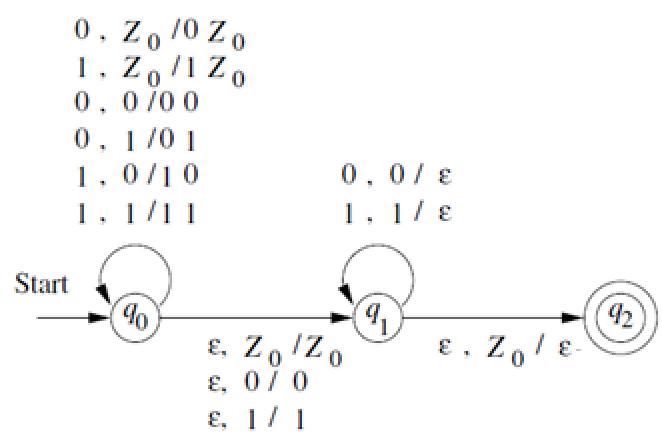


Figure 6.2(a) A modified PDA.

Acceptance by empty stack

Since the set of accepting states is irrelevant, we shall sometimes leave off the last (seventh) component from the specification of a PDA P.

Acceptance by empty stack

Since the set of accepting states is irrelevant, we shall sometimes leave off the last (seventh) component from the specification of a PDA P.

Thus, P can be written as a six-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0)$.

Do we have two types of PDAs?

- One with final states, other with empty stack …

Do we have two types of PDAs?

- One with final states, other with empty stack …
- NO.

Do we have two types of PDAs?

- One with final states, other with empty stack …
- NO.
- But, for any PDA there are two languages (both are CFLs) associated with that PDA.
  - These two (languages) may or may not be same.

Crucial thing is…

- Set of languages accepted by final state is equal to the set of languages accepted by empty stack.
  - Proof of this one is by construction.
- So, power of PDA is same whether recognition happens either by final state or by empty stack.

6.2.3 From Empty Stack to Final State

Theorem 6.9: If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

6.2.3 From Empty Stack to Final State

Theorem 6.9: If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

PROOF: The idea behind the proof is in Fig. 6.4. We use a new symbol X_0, which must not be a symbol of Γ; X_0 is both the start symbol of P_F and a marker on the bottom of the stack that lets us know when P_N has reached an empty stack.

6.2.3 From Empty Stack to Final State

Theorem 6.9: If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

PROOF: The idea behind the proof is in Fig. 6.4. We use a new symbol X_0, which must not be a symbol of Γ; X_0 is both the start symbol of P_F and a marker on the bottom of the stack that lets us know when P_N has reached an empty stack.

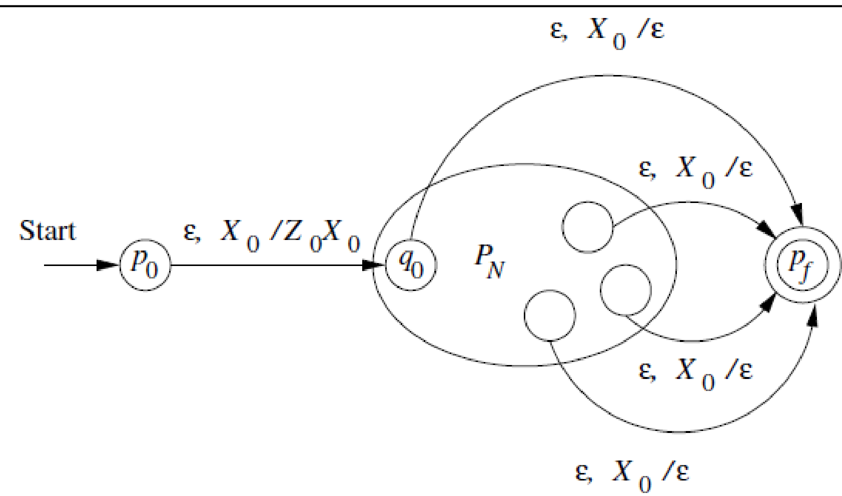


Figure 6.4: P_F simulates P_N and accepts if P_N empties its stack

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

1. $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$. In its start state, P_F makes a spontaneous transition to the start state of P_N, pushing its start symbol Z_0 onto the stack.

2. For all states q in Q, inputs a in Σ or $a = \epsilon$, and stack symbols Y in Γ, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$.

3. In addition to rule (2), $\delta_F(q, \epsilon, X_0)$ contains (p_f, ϵ) for every state q in Q.

6.2.4 From Final State to Empty Stack

Theorem 6.11: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

6.2.4 From Final State to Empty Stack

Theorem 6.11: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

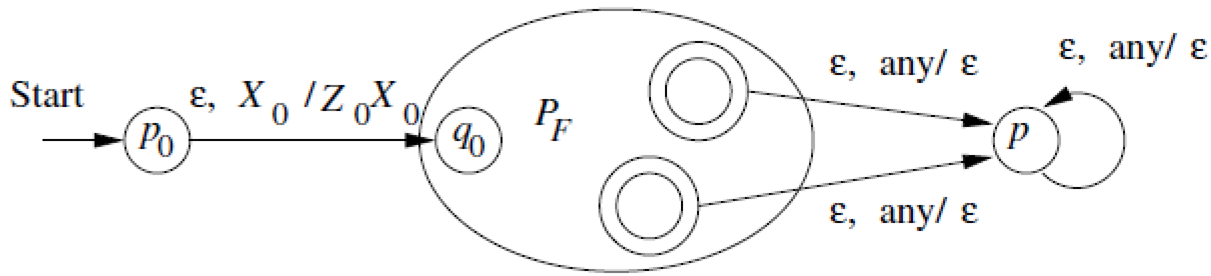


Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

6.2.4 From Final State to Empty Stack

Theorem 6.11: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

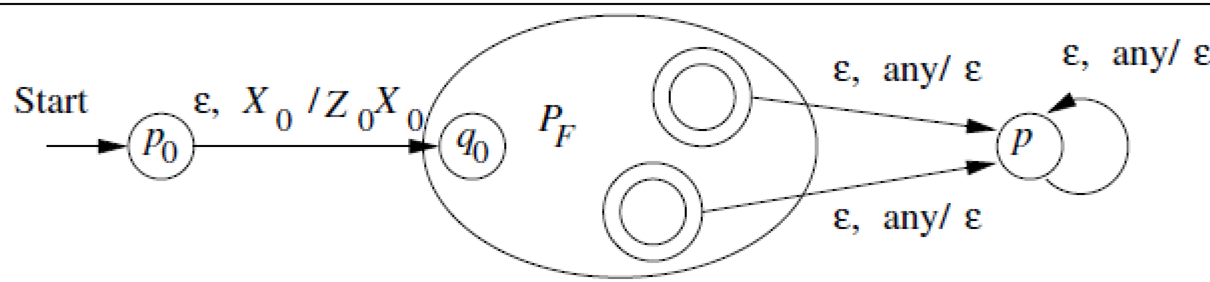


Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

6.2.4 From Final State to Empty Stack

Theorem 6.11: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

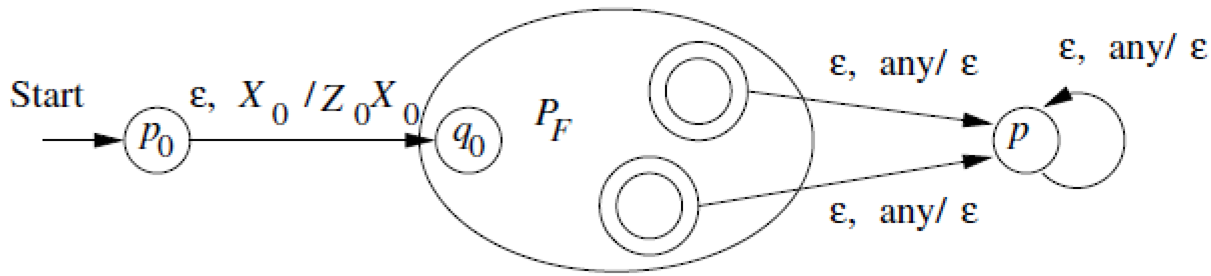


Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

To avoid simulating a situation where P_F accidentally empties its stack without accepting, P_N must also use a marker X_0 on the bottom of its stack.

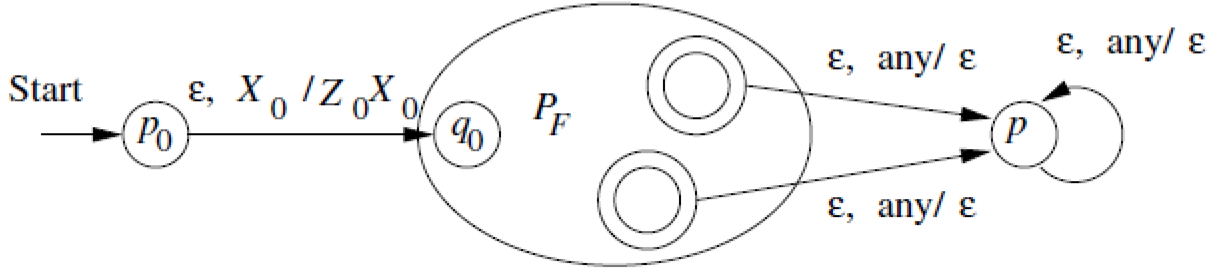


Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$. We start by pushing the start symbol of P_F onto the stack and going to the start state of P_F.

2. For all states q in Q, input symbols a in Σ or $a = \epsilon$, and Y in Γ, $\delta_N(q, a, Y)$ contains every pair that is in $\delta_F(q, a, Y)$. That is, P_N simulates P_F.

3. For all accepting states q in F and stack symbols Y in Γ or $Y = X_0$, $\delta_N(q, \epsilon, Y)$ contains (p, ϵ). By this rule, whenever P_F accepts, P_N can start emptying its stack without consuming any more input.

4. For all stack symbols Y in Γ or $Y = X_0$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$. Once in state p, which only occurs when P_F has accepted, P_N pops every symbol on its stack, until the stack is empty. No further input is consumed.

One point ..

- In P_F if stack becomes empty (in between) then P_F gets stuck. (That PDA gets killed).

One point ..

- In P_F if stack becomes empty (in between) then P_F gets stuck. (That PDA gets killed).
- Now, in this situation P_N has X_0 in the stack, so will this be a problem?

One point ..

- In P_F if stack becomes empty (in between) then P_F gets stuck. (That PDA gets killed).
- Now, in this situation P_N has X_0 in the stack, so will this be a problem?
- If the state is a final state no problem.
- Otherwise, if the state is a non-final one, do we need to do something?

One point ..

- In P_F if stack becomes empty (in between) then P_F gets stuck. (That PDA gets killed).
- Now, in this situation P_N has X_0 in the stack, so will this be a problem?
- If the state is a final state no problem.
- Otherwise, if the state is a non-final one, do we need to do something? NO.
- Since there is no transition (on X_0 being stack top) that PDA gets killed.

Next …

- Equivalence of PDA’s and CFG’s

Next …

- Equivalence of PDA’s and CFG’s

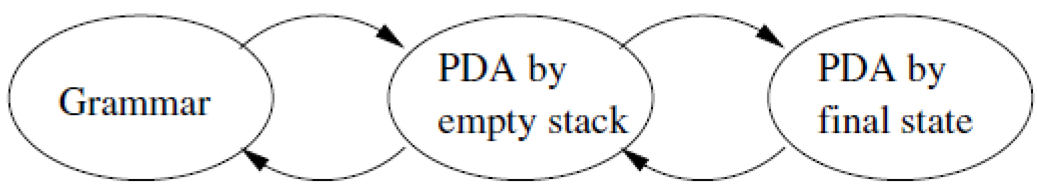


Figure 6.8: Organization of constructions showing equivalence of three ways of defining the CFL’s

CFG to PDA

Let $G = (V, T, Q, S)$ be a CFG.

CFG to PDA

Let $G = (V, T, Q, S)$ be a CFG.

Construct the PDA P that accepts $L(G)$ by empty stack as follows:

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

CFG to PDA

Let $G = \ (V, T, Q, S)$ be a CFG.

Construct the PDA P that accepts $L(G)$ by empty stack as follows:

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

where transition function δ is defined by:

1. For each variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \text{ is a production of } G\}$$

2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$.

Example 6.12: Let us convert the expression grammar of Fig. 5.2 to a PDA. Recall this grammar is:

$$
\begin{aligned}
E &\rightarrow I \mid E * E \mid E + E \mid (E) \\
I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

- Can you identify (V,T,Q,S) of this CFG?

Example 6.12: Let us convert the expression grammar of Fig. 5.2 to a PDA. Recall this grammar is:

$$
\begin{aligned}
E &\rightarrow I \mid E * E \mid E + E \mid (E) \\
I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

- Can you identify (V,T,Q,S) of this CFG?
- $V = \{E, I\}$
- $T = \{\, a, b, 0,1, +,*, (,) \,\}$
- $S = E$

$$E \rightarrow I \mid E * E \mid E + E \mid (E)$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

- For the PDA stack alphabet is $\Gamma = \{E, I, a, b, 0, 1, +, *, (,)\}$
- Start symbol of the stack is E.
- We will have only one state, call it q.

$$
\begin{aligned}
E &\rightarrow I \mid E * E \mid E + E \mid (E) \\
I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

- For the PDA stack alphabet is $\Gamma = \{E, I, a, b, 0, 1, +, *, (,)\}$
- Start symbol of the stack is E.
- We will have only one state, call it q.

$$\delta(q, \epsilon, E) = \{(q, I),\ (q, E + E),\ (q, E * E),\ (q, (E))\}.$$

$$\delta(q, \epsilon, I) = \{(q, a),\ (q, b),\ (q, Ia),\ (q, Ib),\ (q, I0),\ (q, I1)\}.$$

$$E \rightarrow I \mid E * E \mid E + E \mid (E)$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

- For the PDA stack alphabet is $\Gamma = \{E, I, a, b, 0,1, +,*, (,)\}$
- Start symbol of the stack is E.
- We will have only one state, call it q.

$$\delta(q, \epsilon, E) = \{(q, I),\ (q, E + E),\ (q, E * E),\ (q, (E))\}.$$

$$\delta(q, \epsilon, I) = \{(q, a),\ (q, b),\ (q, Ia),\ (q, Ib),\ (q, I0),\ (q, I1)\}.$$

$\delta(q, a, a) = \{(q, \epsilon)\}$; $\delta(q, b, b) = \{(q, \epsilon)\}$; $\delta(q, 0, 0) = \{(q, \epsilon)\}$; $\delta(q, 1, 1) = \{(q, \epsilon)\}$; $\delta(q, (, () = \{(q, \epsilon)\}$; $\delta(q,),)) = \{(q, \epsilon)\}$; $\delta(q, +, +) = \{(q, \epsilon)\}$; $\delta(q, *, *) = \{(q, \epsilon)\}$.

Have you noted?

- The PDA we constructed simulates, which derivation?

Have you noted?

- The PDA we constructed simulates, which derivation?
- It is “left-most derivation”

Have you noted?

- The PDA we constructed simulates, which derivation?
- It is “left-most derivation”
- In compilers, these are top-down parsers
  - LL parsers;
    - but non-determinism is a problem.
    - Backtracking (to try the other choice)
    - Parse table (to find feasible choices; at that time)

Compilers

- We have parsers which are bottom-up
- Which will simulate right-most derivation
- These are called LR parsers.

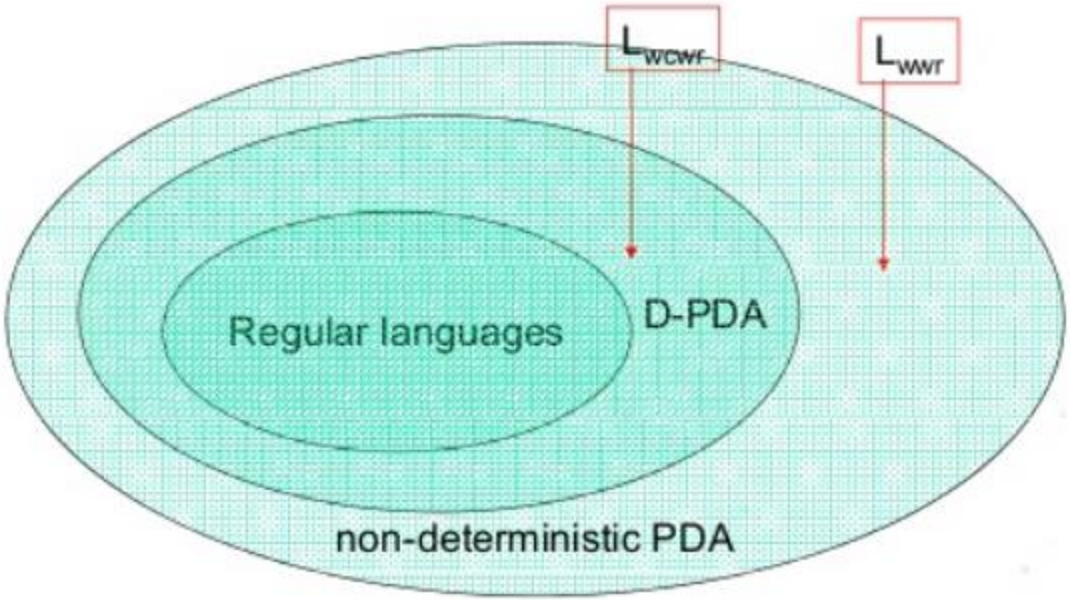
- One notable drawback is all these parsers have their own limitations, and works only for subclasses of CFLs;
  - Some superior, some inferior …

PDA to CFG

- We skip this in this basic course.

Deterministic PDA

- DPDA
- Can recognize a proper subset of CFLs
- Parsers (used in compilers), mostly are DPDAs.
  - Most of our programming languages are in the subclass which can be recognized by DPDAs.



It holds that:

$$\left\{\begin{array}{c}\text{Deterministic}\\ \text{Context-Free}\\ \text{Languages}\\ \text{(DPDA)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{PDAs}\end{array}\right\}$$

Since every DPDA is also a PDA

DPDA

- Remove choice.

DPDA

- Atmost one choice. But ϵ moves (should we remove them?   ).

DPDA

- Atmost one choice. But ϵ moves (should we remove them? No).
- For any $q \in Q, a \in \Sigma, or\ a = \epsilon, and\ X \in \Gamma,$ we have
  1) $|\delta(q, a, X)| \leq 1,$ and
  2) For any a except ϵ,

     $|\delta(q, a, X)| = 1 \quad \Rightarrow \quad |\delta(q, \epsilon, X)| = 0.$

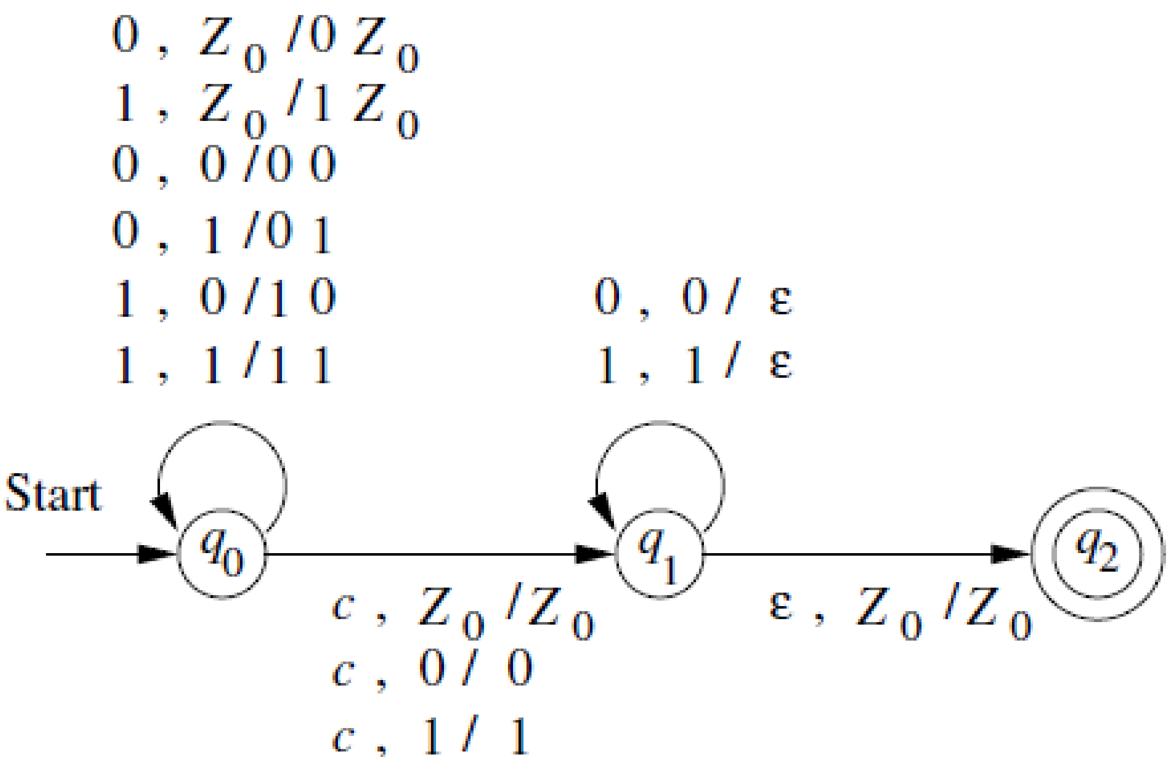


Figure 6.11: A deterministic PDA accepting L_{wcwr}

Regular Languages and DPDAs

Theorem 6.17: If L is a regular language, then $L = L(P)$ for some DPDA P.

Regular Languages and DPDAs

Theorem 6.17: If L is a regular language, then $L = L(P)$ for some DPDA P.

- Proof is simple.
  - DFA is there for the regular language.
  - What about stack??

Regular Languages and DPDAs

Theorem 6.17: If L is a regular language, then $L = L(P)$ for some DPDA P.

- Proof is simple.
  - DFA is there for the regular language.
  - What about stack??
  - Ignore the stack.

Regular Languages and DPDAs

Theorem 6.17: If L is a regular language, then $L = L(P)$ for some DPDA P.

- Proof is simple.
  - DFA is there for the regular language.
  - What about stack??
  - Ignore the stack.

- Just see, the theorem is saying about $L(P)$ only.

Formally,

let $A = (Q, \Sigma, \delta_A, q_0, F)$ be a DFA. Construct DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

Formally,

let $A = (Q, \Sigma, \delta_A, q_0, F)$ be a DFA. Construct DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

If $\delta_A(q, a) = p$ then $\delta_P(q, a, Z_0) = \{(p, Z_0)\}$ for all states p and q in Q.

Formally,

let $A = (Q, \Sigma, \delta_A, q_0, F)$ be a DFA. Construct DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

If $\delta_A(q, a) = p$ then $\delta_P(q, a, Z_0) = \{(p, Z_0)\}$ for all states p and q in Q.

We claim that $(q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, Z_0)$ if and only if $\hat{\delta}_A(q_0, w) = p$.

DPDA and *N(P)* ??

- For some regular languages, there can no DPDA by *empty stack* that recognizes the language.

DPDA and *N(P)* ??

- For some regular languages, there can no DPDA by *empty stack* that recognizes the language.
- But, for a proper subset of regular languages, it is possible to build a DPDA by empty stack.
  - These are characterized by "prefix property".
- DPDA by empty stack can recognize some non-regular languages also, provided they obey the property.

Prefix property

- A language L has the prefix property, if there are no two distinct strings x and y in L such that x is a prefix of y.

Prefix property

- A language L has the prefix property, if there are no two distinct strings x and y in L such that x is a prefix of y.
- L_{wcwr} has this property.

Prefix property

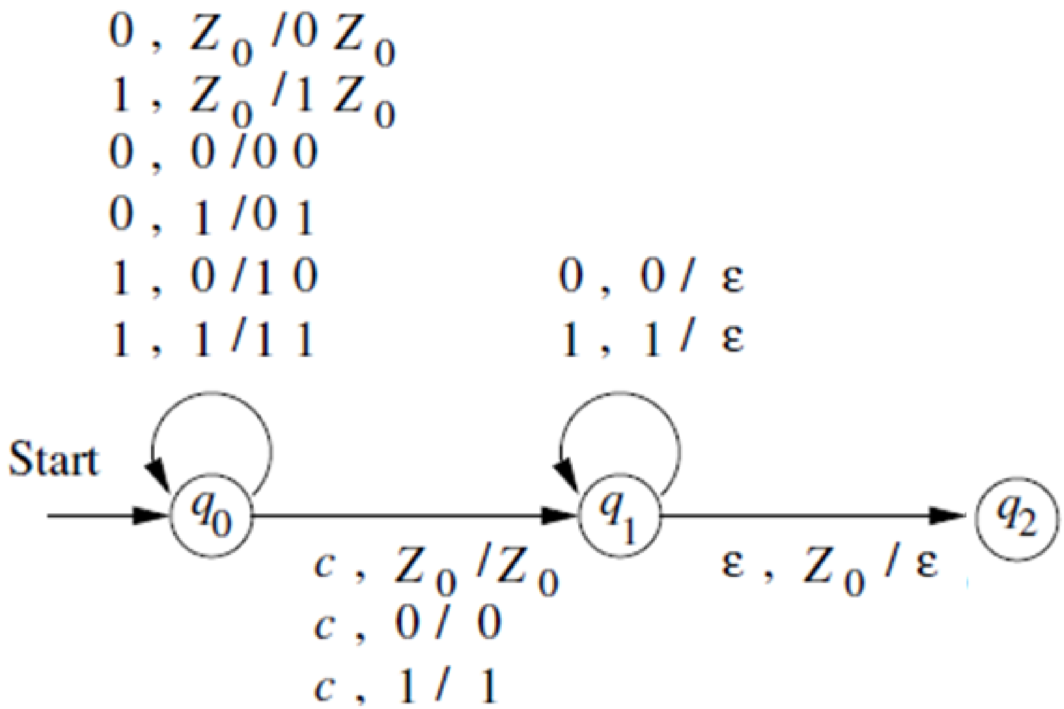
- A language L has the prefix property, if there are no two distinct strings x and y in L such that x is a prefix of y.
- L_{wcwr} has this property.
- 0^* violates this property. See this is regular.

Theorem 6.19: A language L is $N(P)$ for some DPDA P if and only if L has the prefix property and L is $L(P')$ for some DPDA P'. $\square$

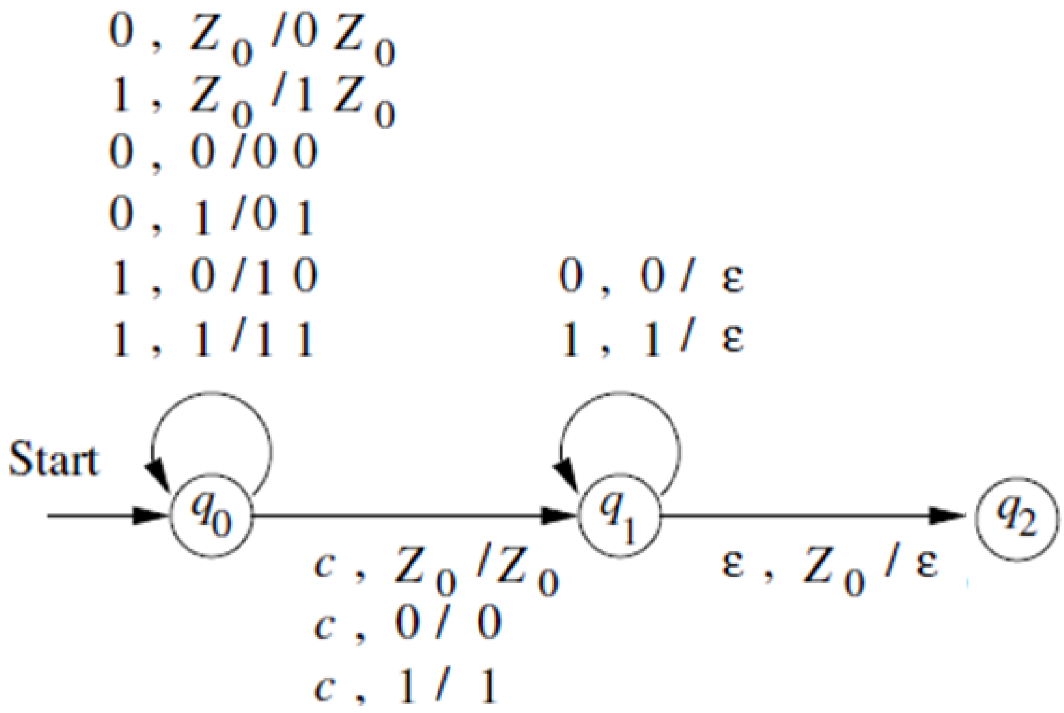
Theorem 6.19: A language L is $N(P)$ for some DPDA P if and only if L has the prefix property and L is $L(P')$ for some DPDA P'. $\square$

See even for 0^* (a regular language) we cannot build a DPDA by empty-stack !!

DPDA s.t. $N(P) = L_{wcwr}$

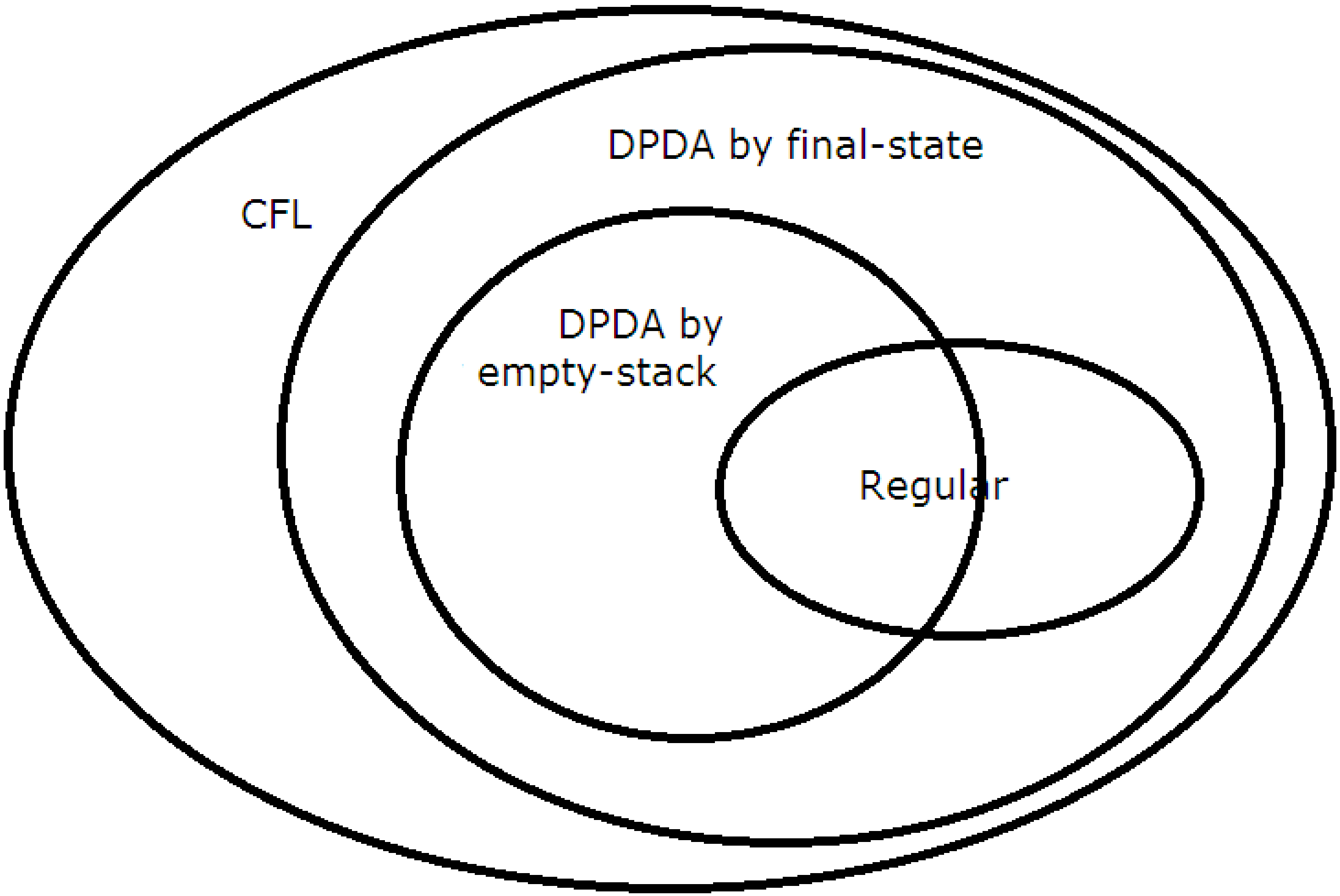


DPDA s.t. $N(P) = L_{wcwr}$



This is not a regular language.

- There is no DPDA to recognize the CFL L_{wwr}
- Proof is complex. But we can see the idea behind..


CFL
DPDA by final-state
DPDA by
empty-stack
Regular

- Some points to note,

Theorem 6.20: If $L = N(P)$ for some DPDA P, then L has an unambiguous context-free grammar.

Theorem 6.21: If $L = L(P)$ for some DPDA P, then L has an unambiguous CFG.

- We cannot make converse of these statements.