Notes on Models of Computation
Chapter 7

H. Conrad Cunningham

06 April 2022

Contents

Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677

(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

Note: These notes were written primarily to accompany my use of Chapter 1 of the Linz textbook *An Introduction to Formal Languages and Automata* [[1].

7 Pushdown Automata

Finite automata cannot recognize all context-free languages.

To recognize language $\{a^n b^n : n \geq 0\}$, an automaton must do more than just verify that all a's precede all b's; it must count an unbounded number of symbols. This cannot be done with the finite memory of a dfa.

To recognize language $\{ww^R : w \in \Sigma^*\}$, an automaton must do more than just count; it must remember a sequence of symbols and compare it to another sequence in reverse order.

Thus, to recognize context-free languages, an automaton needs unbounded memory. The second example above suggests using a stack as the "memory".

Hence, the class of machines we examine in this chapter are called *pushdown automata*.

In this chapter, we examine the connection between context-free languages and pushdown automata.

Unlike the situation with finite automata, deterministic and nondeterministic pushdown automata differ in the languages they accept.

- *Nondeterministic pushdown automata (npda)* accept precisely the class of context-free languages.
- *Deterministic pushdown automata (dpda)* just accept a subset–the *deterministic context-free languages*.

7.1 Nondeterministic Pushdown Automata

7.1.1 Schematic Drawing

Linz Figure 7.1 illustrates a pushdown automaton.

On each move, the control unit reads a symbol from the input file. Based on the input symbol and symbol on the top of the stack, the control unit changes the content of the stack and enters a new state.

7.1.2 Definition of a Pushdown Automaton

Linz Definition 7.1 (Nondeterministic Pushdown Accepter): A *nondeterministic pushdown accepter (npda)* is defined by the tuple

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

where

Q is a finite set of internal states of the control unit,
Σ is the input alphabet,
Γ is a finite set of symbols called the *stack alphabet*,
$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$ is the transition

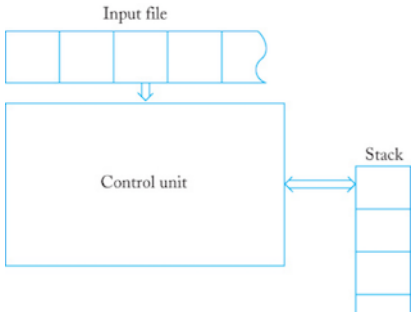


Figure 1: **Linz Fig. 7.1: Pushdown Automaton**

function,
$q_0 \in Q$ is the initial state of the control unit,
$z \in \Gamma$ is the *stack start symbol*,
$F \subseteq Q$ is the set of final states.

Note that the input and stack alphabets may differ and that start stack symbol z must be an element of Γ.

Consider the transition function $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$.

- 1st argument from Q is the current state.
- 2nd argument from $\Sigma \cup \{\lambda\})$ is either the next input symbol or λ for a move that does not consume input.
- 3rd argument from Γ is the current symbol at top of stack. (The stack cannot be empty! The stack start symbol represents the empty stack.)
- The result value is a finite set of pairs (q, w) where
  - q is the new state,
  - w is the (possibly empty) string that replaces the top symbol on the stack. (The first element of w will be the new top of the stack, second element under that, etc.)

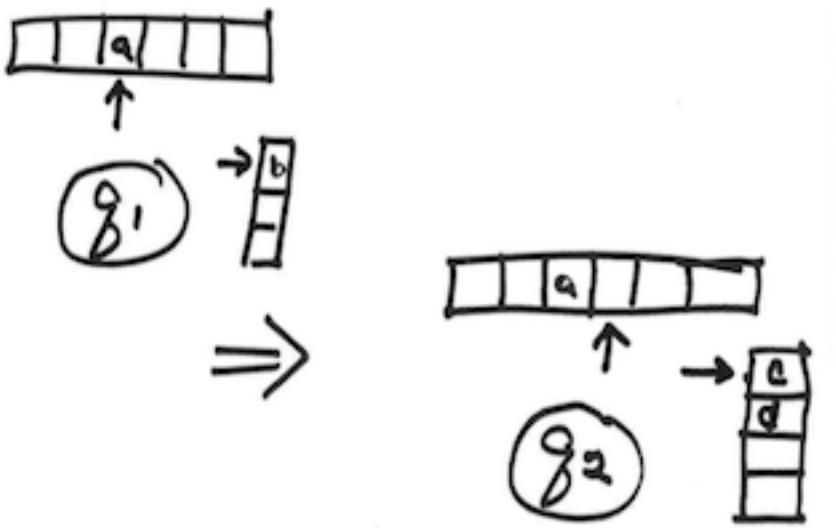
The machine is nondeterministic, but there are only a finite number of possible results for each step.

7.1.3 Linz Example 7.1

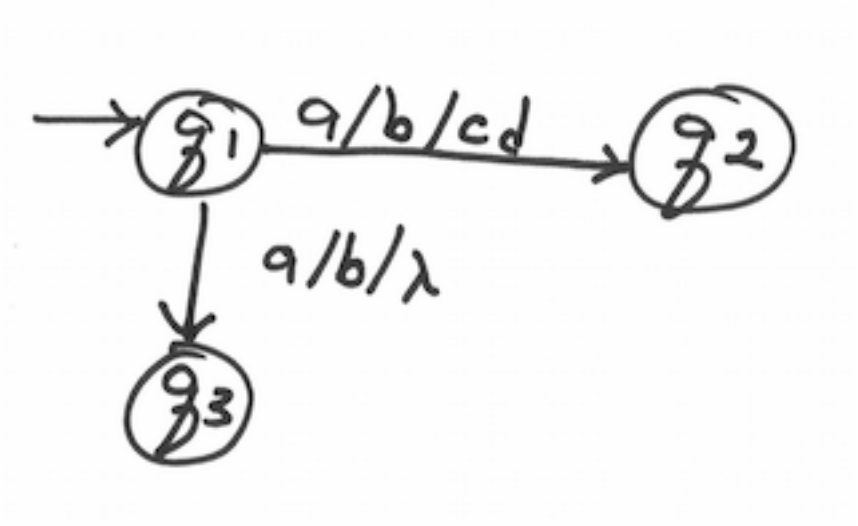
Suppose the set of transition rules of an npda contains

$$\delta(q_1, a, b) = \{(q_2, cd), (q_3, \lambda)\}.$$

A possible change in the state of the automaton from q_1 to q_2 is shown in following diagram.



This transition function can be drawn as a transition graph as shown below. The triple marking the edges represent the input symbol, the symbol at the top of the stack, and the string pushed back on the stack. Here we use "/" to separate the elements of the triple on the edges; the book uses commas for this purpose.



7.1.4 Linz Example 7.2

Consider an npda $(Q, \Sigma, \Gamma, \delta, q_0, z, F)$ where

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$

$\Gamma = \{0, 1\}$
$z = 0$
$F = \{q_3\}$

with the initial state q_0 and the transition function defined as follows:

1. $\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \lambda)\}$
2. $\delta(q_0, \lambda, 0) = \{(q_3, \lambda)\}$
3. $\delta(q_1, a, 1) = \{(q_1, 11)\}$
4. $\delta(q_1, b, 1) = \{(q_2, \lambda)\}$
5. $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$
6. $\delta(q_2, \lambda, 0) = \{(q_3, \lambda)\}$.

There are no transitions defined for final state q_3 or in the cases

$(q_0, b, 0)$, $(q_2, a, 1)$, $(q_0, a, 1)$, $(q_0, b, 1)$, $(q_1, a, 0)$, and $(q_1, b, 0)$.

These are *dead configurations* of the npda. In these cases, δ maps the arguments to the empty set.

The machine executes as follows.

1. The first transition rule is nondeterministic, with two choices for input symbol a and stack top 0.
   (a) The machine can push 1 on the stack and transition to state q_1. (This is the only choice that will allow the machine to accept additional input.)
   (b) The machine can pop the start symbol 0 and transition to final state q_3. (This is the only choice that will allow the machine to accept a single a.)
2. For stack top z, the machine can also transition from the initial state q_0 to final state q_3 without consuming any input. (This is only choice that will allow the machine to accept an empty string.) Note that rule 2 overlaps with rule 1, giving additional nondeterminism.
3. While the machine reads a's, it pushes a 1 on the stack and stays in state q_1.
4. When the machine reads a b (with stack top 1), it pops the 1 from the stack and transitions to state q_2.
5. While the machine reads b's (with stack top 1), it pops the 1 from the stack and stays in state q_2.
6. When the machine encounters the stack top 0, it pops the stack and transitions to final state q_3.

Acceptance or rejection?

- If the machine reaches final state q_3 with *no unprocessed input* using any possible sequence of transitions, then the machine *accepts* the input string.
- If every sequence of possible transitions reaches a configuration in which no move is defined or reaches the final state with unprocessed input remaining, then the machine *rejects* the input string.

The machine accepts:

- λ (via rule 2)
- singleton string a (via rule 1b)
- any string in which there are some number of a's followed by the same number of b's (via rules 1a-3-4-5-6 as applicable)

Other strings will always end in dead configurations. For example:

- b gets stuck in q_3 with unprocessed input (via rule 2)
- aa gets stuck in q_1 (via rules 1a-3) or in q_3 with unprocessed input (via rule 1b or rule 2)
- aab gets stuck in q_2 with stack top 1 (via rules 1a-3-4) or in q_3 with unprocessed input (via rule 1b or 2)
- abb gets stuck in q_3 with unprocessed input (via rule 1b or rule 2 or rules 1a-4-5-6)
- aba gets stuck in q_2 (via rules 1a-4) or in q_3 with unprocessed input (via rule 1b or rule 2 or rules 1a-4-6)

Thus, it is not difficult to see that $L = \{a^n b^n : n \geq 0\} \cup \{a\}$.

Linz Figure 7.2 shows a transition graph for this npda. The triples marking the edges represent the input symbol, the symbol at the top of the stack, and the string pushed back on the stack.

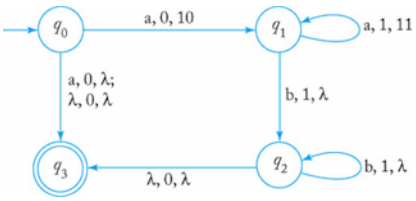


Figure 2: **Linz Fig. 7.2: Transition Graph for Example 7.2**

7.1.5 Instantaneous Descriptions of Pushdown Automata

Transition graphs are useful for describing pushdown automata, but they are not useful in formal reasoning about them. For that, we use a concise notation for describing configurations of pushdown automata with tuples.

The triple (q, w, u) where

q is the control unit state
w is the unread input string
u is the stack as string, beginning with the top symbol

is called an *instantaneous description* of a pushdown automaton.

We introduce the symbol $\vdash$ to denote a *move* from one instantaneous description to another such that

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

is possible if and only if

$$(q_2, y) \in \delta(q_1, a, b).$$

We also introduce the notation $\vdash^*$ to denote an arbitrary number of steps of the machine.

7.1.6 Language Accepted by an NPDA

Linz Definition 7.2 (Language Accepted by a Pushdown Automaton): Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ be a nondeterministic pushdown automaton. The language accepted by M is the set

$$L(M) = \{w \in \Sigma^* : (q_0, w, z) \vdash_M^* (p, \lambda, u), p \in F, u \in \Gamma^*\}.$$

In words, the language accepted by M is the set of all strings that can put M into a final state at the end of the string. The stack content u is irrelevant to this definition of acceptance.

7.1.7 Linz Example 7.4

Construct an npda for the language

$$L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}.$$

We must count a's and b's, but their relative order is not important.

The basic idea is:

- on "a", push 0
- on "b", pop 0

But, what if $n_b > n_a$ at some point?

We must allow a "negative" count. So we modify the above solution as follows:

- on "a",

  if top is z or 0
  push 0
  else if top is 1
  pop 1

- on "b",

  if top is z or 1
  push 1
  else if top is 0
  pop 0

So the solution is an npda.

$M = (\{q_0, q_f\}, \{a, b\}, \{0, 1, z\}, \delta, q_0, z, \{q_f\})$, with δ given as

1. $\delta(q_0, \lambda, z) = \{(q_f, z)\}$
2. $\delta(q_0, a, z) = \{(q_0, 0z)\}$
3. $\delta(q_0, b, z) = \{(q_0, 1z)\}$
4. $\delta(q_0, a, 0) = \{(q_0, 00)\}$
5. $\delta(q_0, b, 0) = \{(q_0, \lambda)\}$
6. $\delta(q_0, a, 1) = \{(q_0, \lambda)\}$
7. $\delta(q_0, b, 1) = \{(q_0, 11)\}$.

Linz Figure 7.3 shows a transition graph for this npda.

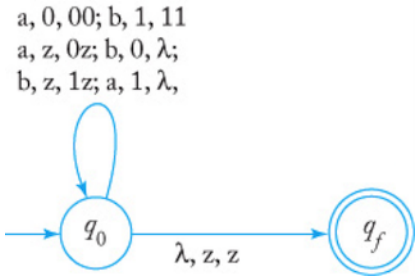


Figure 3: **Linz Fig. 7.3: Transition Graph for Example 7.4**

In processing the string *baab*, the npda makes the following moves (as indicated by transition rule number):

		$(q_0, baab, z)$
(3)	$\vdash$	$(q_0, aab, 1z)$
(6)	$\vdash$	(q_0, ab, z)
(2)	$\vdash$	$(q_0, b, 0z)$
(5)	$\vdash$	(q_0, λ, z)
(1)	$\vdash$	(q_f, λ, z)

Hence, the string is accepted.

7.1.8 Linz Example 7.5

Construct an npda for accepting the language $L = \{ww^R : w \in \{a, b\}^+\}$,

The basic idea is:

- push symbols from w on stack from left to right
- pop symbols from stack for w^R (which is w right-to-left)

Problem: How do we find the middle?

Solution: Use nondeterminism!

- Each symbol could be at middle.

- Automaton "guesses" when to switch.

For $L = \{ww^R : w \in \{a, b\}^+\}$, a solution to the problem is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$, where:

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, z\}$ – which is Σ plus the staack start symbol $F = \{q_2\}$

The transition function can be visualized as having several parts.

- a set of transitions to push w on the stack (one for each element of $\Sigma \times \Gamma$):

1. $\delta(q_0, a, a) = \{(q_0, aa)\}$
2. $\delta(q_0, b, a) = \{(q_0, ba)\}$
3. $\delta(q_0, a, b) = \{(q_0, ab)\}$
4. $\delta(q_0, b, b) = \{(q_0, bb)\}$
5. $\delta(q_0, a, z) = \{(q_0, az)\}$
6. $\delta(q_0, b, z) = \{(q_0, bz)\}$

- a set of transitions to guess the middle of the string, where the npda switches from state q_0 to q_1 (any position is potentially the middle):

7. $\delta(q_0, \lambda, a) = \{(q_1, a)\}$
8. $\delta(q_0, \lambda, b) = \{(q_1, b)\}$

- a set of transitions to match w^R against the contents of the stack:

9. $\delta(q_1, a, a) = \{(q_1, \lambda)\}$
10. $\delta(q_1, b, b) = \{(q_1, \lambda)\}$

- a transition to recognize a successful match:

11. $\delta(q_1, \lambda, z) = \{(q_2, z)\}$

Remember that, to be accepted, a final state must be reached with no unprocessed input remaining.

The sequence of moves accepting *abba* is as follows, where the number in parenthesis gives the transition rule applied:

		$(q_0, abba, z)$
(5)	$\vdash$	(q_0, bba, az)
(2)	$\vdash$	(q_0, ba, baz)
(8)	$\vdash$	(q_1, ba, baz)
(10)	$\vdash$	(q_1, a, az)
(9)	$\vdash$	(q_1, λ, z)
(11)	$\vdash$	(q_2, z)

7.2 Pushdown Automata and Context-Free Languages

7.2.1 Pushdown Automata for CFGs

Underlying idea: Given a context-free language, construct an npda that simulates a leftmost derivation for any string in the language.

We assume the context-free language is represented as grammar in *Greibach Normal Form*, as defined in Linz Chapter 6. We did not cover that chapter, but the definition and key theorem are shown below.

Greibach Normal Form restricts the positions at which terminals and variables can appear in a grammar's productions.

Linz Definition 6.5 (Greibach Normal Form): A context-free grammar is said to be in *Greibach Normal Form* if all productions have the form

$$A \to ax,$$

where $a \in T$ and $x \in V^*$.

The structure of a grammar in Greibach Normal Form is similar to that of an s-grammar except that, unlike s-grammars, the grammar does not restrict pairs (A, a) to single occurrences within the set of productions.

Linz Theorem 6.7 (Existence of Greibach Normal Form Grammars): For every context-free grammar G with $\lambda \notin L(G)$, there exists an equivalent grammar $\hat{G}$ in Greibach normal form.

Underlying idea, continued: Consider a sentential form, for example,

$$x_1x_2x_3x_4x_5x_6$$

where $x_1x_2x_3$ are the *terminals* read from the input and $x_4x_5x_6$ are the variables on the stack.

Consider a production $A \to ax$.

If variable A is on top of stack and terminal a is the next input symbol, then remove A from the stack and push back x.

An npda transition function δ for $A \to ax$ must be defined to have the move

$$(q, aw, Ay) \vdash (q, w, xy)$$

for some state q, input string suffix w, and stack y. This, we define δ such that

$$\delta(q, a, A) = \{(q, x)\}.$$

7.2.2 Linz Example 7.6

Construct a pda to accept the language generated by grammar with productions

$$S \to aSbb \mid a.$$

First, we transform this grammar into Greibach Normal Form:

$S \to aSA \mid a$
$A \to bB$
$B \to b$

We define the pda to have three states – an initial state q_0, a final state q_2, and an intermediate state q_1.

We define the initial transition rule to push the start symbol S on the stack:

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$$

We simulate the production $S \to aSA$ with a transition that reads a from the input and replaces S on the top of the stack by SA.

Similarly, we simulate the production $S \to a$ with a transition that reads a while simply removing S from the top of the stack. We represent these two productions in the pda as the nondeterministic transition rule:

$$\delta(q_1, a, S) = \{(q_1, SA), (q_1, \lambda)\}$$

Doing the same for the other productions, we get transition rules:

$\delta(q_1, b, A) = \{(q_1, B)\}$
$\delta(q_1, b, B) = \{(q_1, \lambda)\}$

When the stack start symbol appears at the top of the stack, the derivation is complete. We define a transition rule to move the pda into its final state:

$$\delta(q_1, \lambda, z) = \{(q_2, \lambda)\}$$

7.2.3 Constructing an NPDA for a CFG

Linz Theorem 7.1 (Existence of NPDA for Context-Free Language): For any context-free language L, there exists an npda M such that $L = L(M)$.

Proof: The proof partly follows from the following construction (algorithm).

Algorithm to construct an npda for a context-free grammar

- Let $G = (V, T, S, P)$ be a grammar for L in Greibach Normal Form.
- Construct npda $M = (\{q_0, q_1, q_f\}, T, V \cup \{z\}, \delta, q_0, z, \{q_f\})$ where:
  - $z \notin V$
  - T is the input alphabet for the npda
  - $V \cup \{z\}$ is the stack alphabet for the npda
- Define transition rule $\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$ to initialize the stack.
- For every $A \to au$ in P, define transition rules

  $$(q_1, u) \in \delta(q_1, a, A)$$

  that read a, pop A, and push u. (Note the possible nondeterminism.)
- Define transition rule $\delta(q_1, \lambda, z) = \{(q_f, z)\}$ to detect the end of processing.

7.2.4 Linz Example 7.7

Consider the grammar:

$S \to aA$
$A \to aABC \mid bB \mid a$
$B \to b$
$C \to c$

This grammar is already in Greibach Normal Form. So we can apply the algorithm above directly.

In addition to the transition rules for the startup and shutdown, i.e.,

1. $\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$
2. $\delta(q_1, \lambda, z) = \{(q_f, z)\}$

the npda has the following transition rules for the productions:

3. $\delta(q_1, a, S) = \{(q_1, A)\}$
4. $\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$
5. $\delta(q_1, b, A) = \{(q_1, B)\}$
6. $\delta(q_1, b, B) = \{(q_1, \lambda)\}$
7. $\delta(q_1, c, C) = \{(q_1, \lambda)\}$

The sequence of moves made by M in processing is $aaabc$ is

		$(q_0, aaabc, z)$
(1)	$\vdash$	$(q_1, aaabc, Sz)$
(3)	$\vdash$	$(q_1, aabc, Az)$
(4a)	$\vdash$	$(q_1, abc, ABCz)$
(4b)	$\vdash$	(q_1, bc, BCz)
(6)	$\vdash$	(q_1, c, Cz)
(7)	$\vdash$	(q_1, λ, z)
(2)	$\vdash$	(q_f, λ, z)

This corresponds to the derivation

$S \Rightarrow aA \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabC \Rightarrow aaabc.$

The previous construction assumed Greibach Normal Form. This is not necessary, but the needed construction technique is more complex, as sketched below.

$A \to Bx$

$(q_1, Bx) \in \delta(q_1, \lambda, A)$

$A \to abCx$

e.g.,
$(q_2, \lambda) \in \delta(q_1, a, a)$

$(q_3, \lambda) \in \delta(q_2, b, b)$
$(q_1, Cx) \in \delta(q_3, \lambda, A)$

etc.

7.2.5 Constructing a CFG for an NPDA

Linz Theorem 7.2 (Existence of a Context-Free Language for an NPDA): If $L = L(M)$ for some npda M, then L is a context-free language.

Basic idea: To construct a context-free grammar from an npda, reverse the previous construction.

That is, construct a grammar to simulate npda moves:

- The stack content becomes the variable part of the grammar.
- The processed input becomes the terminal prefix of sentential form.

This leads to a relatively complicated construction. This is described in the Linz textbook in more detail, but we will not cover it in this course.

7.3 Deterministic Pushdown Automata and Deterministic Context-Free Languages

7.3.1 Deterministic Pushdown Automata

A *deterministic pushdown accepter (dpda)* is a pushdown automaton that never has a choice in its move.

Linz Definition 7.3 (Deterministic Pushdown Automaton): A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ is *deterministic* if it is an automaton as defined in Linz Definition 7.1, subject to the restrictions that, for every $q \in Q, a \in \Sigma \cup \{\lambda\}$, and $b \in \Gamma$,

1. $\delta(q, a, b)$ contains at most one element,
2. if $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$.

Restriction 1 requires that for, any given input symbol and any stack top, at most one move can be made.

Restriction 2 requires that, when a λ-move is possible for some configuration, no input-consuming alternative is available.

Consider the difference between this dpda definition and the dfa definition:

- A dpda allows λ-moves, but the moves are deterministic.
- A dpda may have dead configurations.

Linz Definition 7.4 (Deterministic Context-Free Language): A language L is a *deterministic context-free language* if and only if there exists a dpda M such that $L = L(M)$.

7.3.2 Linz Example 7.10

The language $L = \{a^n b^n : n \geq 0\}$ is a deterministic context-free language.

The pda $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{0, 1\}, \delta, q_0, 0, \{q_0\})$ with transition rules

$\delta(q_0, a, 0) = \{(q_1, 10)\}$
$\delta(q_1, a, 1) = \{(q_1, 11)\}$
$\delta(q_1, b, 1) = \{(q_2, \lambda)\}$
$\delta(q_2, b, 1) = \{(q_2, \lambda)\}$
$\delta(q_2, \lambda, 0) = \{(q_0, \lambda)\}$

accepts the given language. This grammar satisfies the conditions of Linz Definition 7.4. Therefore, it is deterministic.

7.3.3 Linz Example 7.5 Revisited

Consider language

$$L = \{ww^R : w \in \{a, b\}^+\}$$

and machine

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

where:

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, z\}$
$F = \{q_2\}$

The transition function can be visualized as having several parts:

- a set of transition rules to push w on the stack

  $\delta(q_0, a, a) = \{(q_0, aa)\} \leftarrow$ **Restriction 2 violation**
  $\delta(q_0, b, a) = \{(q_0, ba)\}$
  $\delta(q_0, a, b) = \{(q_0, ab)\}$
  $\delta(q_0, b, b) = \{(q_0, bb)\}$
  $\delta(q_0, a, z) = \{(q_0, az)\}$
  $\delta(q_0, b, z) = \{(q_0, bz)\}$

- a set of transition rules to guess the middle of the string, where the npda switches from state q_0 to q_1

  $\delta(q_0, \lambda, a) = \{(q_1, a)\} \leftarrow$ **Restriction 2 violation**
  $\delta(q_0, \lambda, b) = \{(q_1, b)\}$

- a set of transition rules to match w^R against the contents of the stack

  $\delta(q_1, a, a) = \{(q_1, \lambda)\}$
  $\delta(q_1, b, b) = \{(q_1, \lambda)\}$

- a transition rule to recognize a successful match

  $\delta(q_1, \lambda, z) = \{(q_2, z)\}$

This machines violates *Restriciton 2* of Linz Definition 7.3 (Deterministic Push-down Automaton) as indicated above. Thus, it is not deterministic.

Moreover, L is itself not deterministic (which is not proven here).

7.4 Grammars for Deterministic Context-Free Grammars

Deterministic context-free languages are important because they can be parsed efficiently.

- The dpda essentially defines a parsing machine.
- Because it is deterministic, there is no backtracking involved.
- We can thus easily write a reasonably efficient computer program to implement the parser.
- Thus deterministic context-free languages are important in the theory and design of compilers for programming languages.

An *LL-grammar* is a generalization of the concept of s-grammar. This family of grammars generates the deterministic context-free languages.

Compilers for practical programming languages may use top-down parsers based on LL-grammars to parse the languages efficiently.

7.5 References

[1] Peter Linz. 2011. *Formal languages and automata* (Fifth ed.). Jones & Bartlett, Burlington, Massachusetts, USA.