Copyright Warning

Curtin
University of Technology

Theoretical Foundations of Computer Science 300

Lecture 5

Push-Down Automata

Curtin University of Technology

Outline

- What is a Push Down Automaton (PDA)?
  - ➢ Formal definition
  - ➢ State diagram
  - ➢ Examples
- Context Free Languages (CFG)
- Equivalence of PDA and CFG
  - ➢ How to convert a CFG to a PDA
- Non-context free languages
  - ➢ Pumping lemma for CFG

Curtin University of Technology

Unit Learning Outcomes

- Synthesize FA, PDA, CFG, and TMs with specific properties, and convert from one form to another.

Assessment Criteria

- **Model** a specification expressed in English or Mathematics as a PDA.
- **Explain** the operation of a machine on an input string.
- **Express** an English or Mathematical specification as a CFG.
- **Classify** a problem as belonging to the class of PDA and CFG.

PUSHDOWN AUTOMATON

Concept
Formal Definition
Computation

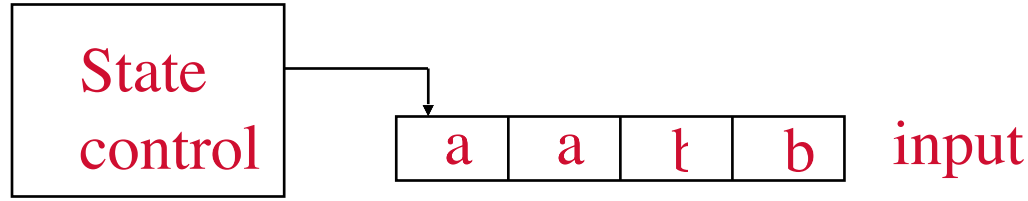
Curtin
University of Technology

Pushdown automata

- Like NFA but with a stack added
  - ➢ Stack provides additional memory
  - ➢ Stack allows recognition of some non-regular languages

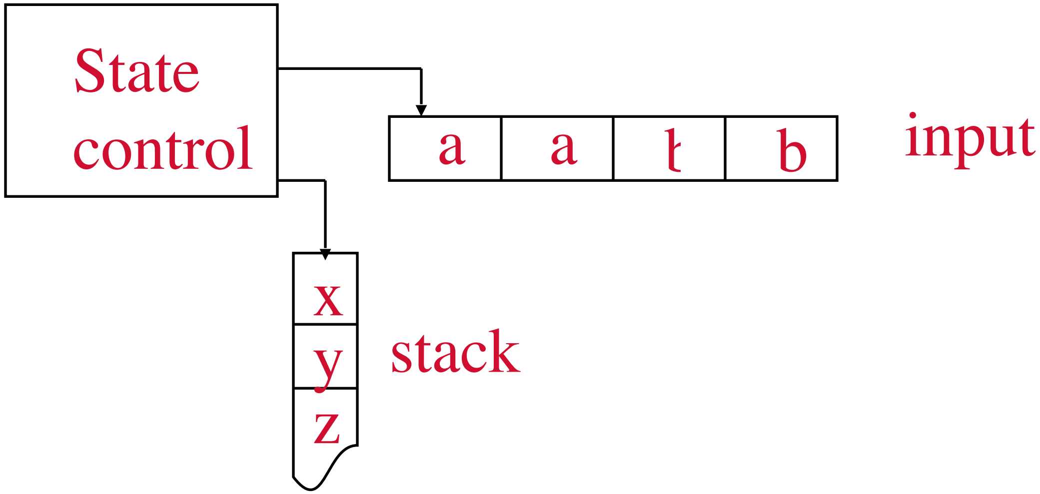
- Note that deterministic PDAs exist
  - ➢ Unlike with DFAs and NFAs, deterministic PDAs are **NOT** equivalent to non-deterministic PDAs
  - ➢ Theoretically these are the next step, but not a useful one
  - ➢ For this unit, assume all PDAs can be non-deterministic.

Schematic of a Finite Automaton



- State control represents the states and transition function
- Tape contains the input string
- Arrow represents the input head
  - ➢ pointing at the next input symbol to be read
- Addition of a stack component will give us the schematic of a pushdown automaton

Schematic of a pushdown automaton



- PDA can write symbols on the stack and read them back later
  - push: writing a symbol at the top of the stack
  - pop: removing a symbol from the top
  - stack can hold unlimited amount of information

PDA

- Example: $L(G) = \{0^n 1^n \mid n \geq 0\}$
  - ➢ Finite automata cannot recognise this language
  - ➢ PDA can store the 0s it has seen, then pop the 0s one by one as 1s are seen
  - ➢ if the stack becomes empty exactly when the input of 1s is finished then accept, otherwise reject the input

- PDA can be non-deterministic
  - ➢ unlike finite automata, non-determinism adds power to the PDA

Formal definition of PDA

- A PDA is a 6-tuple (Q, Σ, Γ, δ, q_0, F), where
  - ➢ Q is the set of states
  - ➢ Σ is the input alphabet
  - ➢ Γ is the stack alphabet
  - ➢ $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(\mathrm{Q} \times \Gamma_\varepsilon)$ is the transition function
  - ➢ $q_0 \in Q$ is the start state, and
  - ➢ $F \in Q$ is the set of accept states
- Note that the problem defines the input alphabet (as usual), but the designer defines the stack alphabet.

Curtin University of Technology

Input and Stack Alphabets of PDA

- Definition of PDA is similar to that of FA except for the addition of a stack
- Stack contains symbols from a stack alphabet Γ
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

Curtin University of Technology

Transition Function of PDA

- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$
- Domain of the transition function is $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$
  - ➢ Current state, next input symbol and top symbol of stack
  - ➢ Either symbol can be ε causing the machine to move without reading input or top of stack
- Range of the transition function is $P(Q \times \Gamma_\varepsilon)$
  - ➢ The machine may enter some new state or possibly write a symbol on the stack
  - ➢ Because of non-determinism, there may several legal moves

Curtin University of Technology

How a PDA Computes

- $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w
  - ➢ if $w = w_1\, w_2 \ldots w_n$, is a string containing members of Σ_ε and
  - ➢ a sequence of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist
  - ➢ that satisfy the following three conditions (on next slide)

Curtin University of Technology

How a PDA Computes

- *M* starts in the start state with an empty stack:
  - $r_0 = q_0$ and $s_0 = \varepsilon$.
- *M* proceeds according to the state, stack and the next input symbol of the transition function:
  - For $i = 0, \ldots, m - 1$, we have $(r_{i+1}, b) \in (r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a,b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$.
- At the end of the input, *M* is in an accept state:
  - $r_m \in F$.

Curtin University of Technology

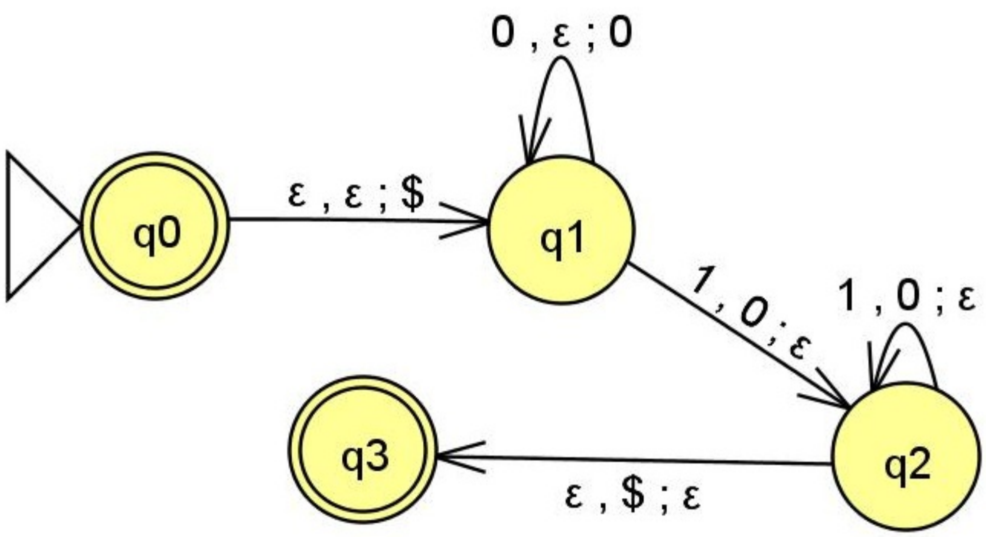
EXAMPLE

Example
State Diagram and Transition Table
Further Examples

Curtin University of Technology

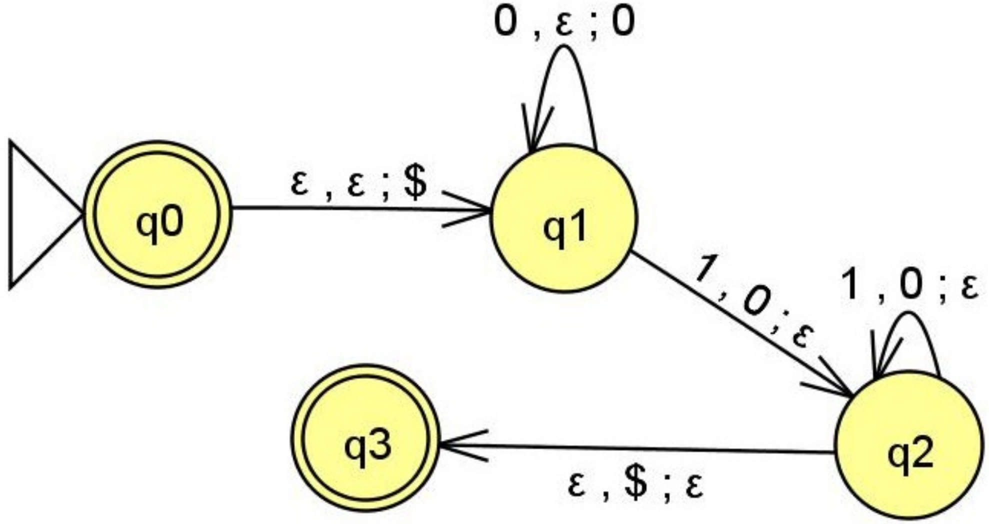
State diagram for a PDA

- PDA recognises $\{0^n 1^n \mid n \geq 0\}$
- $a,b \rightarrow c$ signifies
  - when reading an a from input, replace b at the top of the stack with a c
  - any of $a,b,$ and c may be ε
  - if a is ε, the machine may take this transition without reading from input
  - $ indicates empty stack



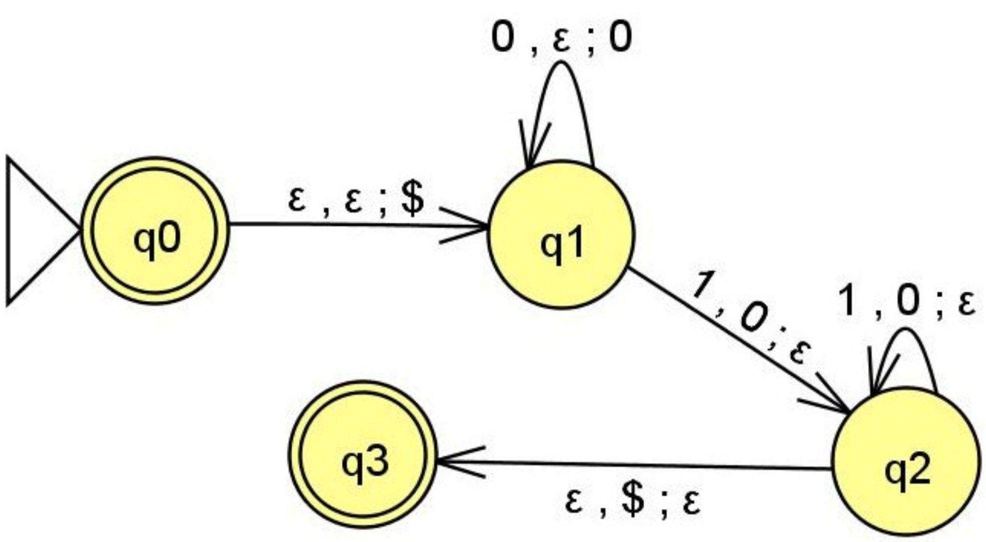
State diagram for a PDA

- ε,ε→$
  - ➢ Without reading from input and the stack, makes the stack empty
- 0,ε→0
  - ➢ On reading a 0 from input write a 0 to the top of stack (without changing whatever else was on there)
- 1,0→ε
  - ➢ When a 1 is read from input and a 0 popped from stack, do not write anything to stack
- ε,$→ ε
  - ➢ When input and stack are empty, do nothing other than change states



Formal description of a PDA

- $M_1=(Q, \Sigma, \Gamma, \delta, q_0, F)$
- $Q = \{q_1, q_2, q_3, q_4\}, \Sigma = \{0,1\}$
- $\Gamma = \{0,\$\}, F = \{q_1, q_4\}$
- δ is given by a state transition table (next page)
- In the formal definition, there is no explicit mechanism to test for an empty stack
  - ➢ hence the use of $

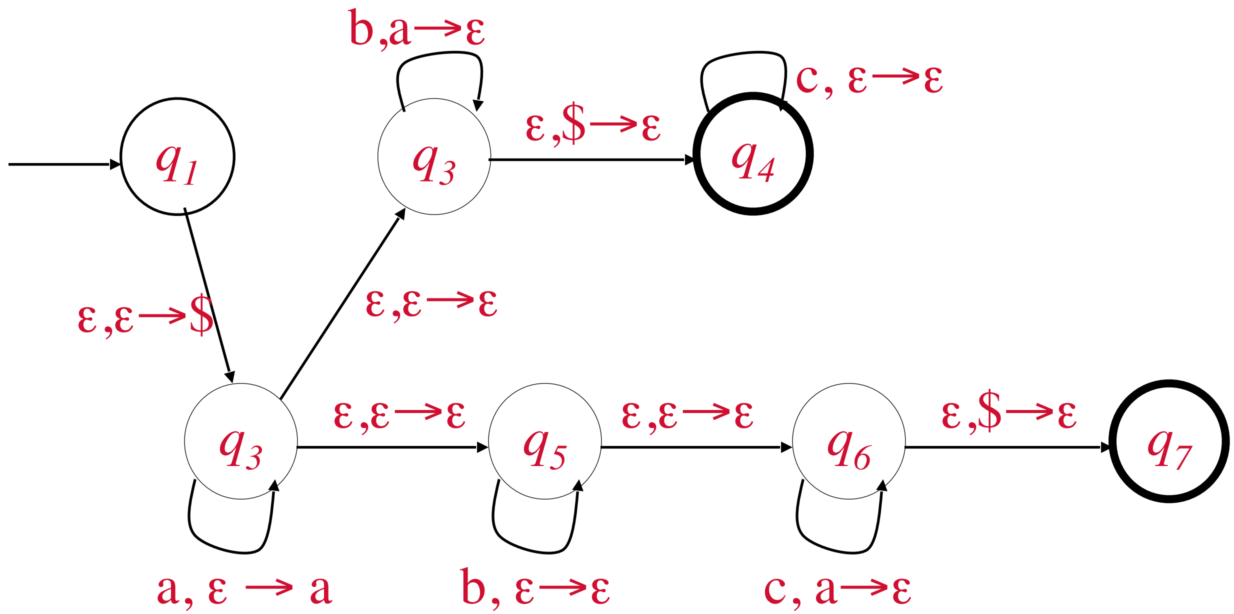


State Transition Table

Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_0									$\{(q_1,\$)\}$
q_1			$\{(q_1,0)\}$	$\{(q_2, \varepsilon)\}$					
q_2				$\{(q_2, \varepsilon)\}$				$\{(q_3, \varepsilon)\}$	
q_3									

Curtin University of Technology

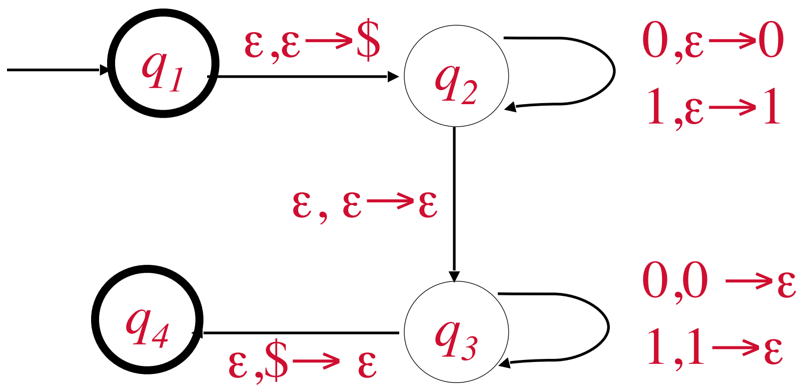
Another PDA Example



- PDA to recognize $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i=j \text{ or } i=k\}$
  - ➢ First read and push the *a*'s
  - ➢ Not known whether to match *b*'s or *c*'s
  - ➢ Use nondeterminism to guess whether to match *b*'s or *c*'s
  - ➢ Nondeterminism essential for recognizing this language

Curtin University of Technology

Another PDA



- PDA recognizing the language $\{ww^R | w \in \{0,1\}^*\}$
  - ➢ Begin by pushing symbols to stack
  - ➢ Non-deterministically guess when the middle of the string is reached and then change to popping symbols
  - ➢ Accept if the stack empties at the same time as the end of input; otherwise reject.

PDAs

- Equivalent in power to Context Free Languages
  - ➢ gives two options for proving a language context free
  - ➢ some languages are more easily described in terms of generators,
  - ➢ others are easier to describe using recognizers

CONTEXT-FREE LANGUAGES

Outline

- Context-free grammars and languages
- Design techniques
- Ambiguity
- Chomsky normal form
  - ➢ CNF theorem

GRAMMARS

Aims & Background
Context Free Grammars (CFG)
Language of a Grammar
Formal Definition

Aims

- To extend our concept of a machine
  - ➢ Studied DFA/NFA
  - ➢ Found associated language (RL)
    - – Characterised as Regular Expressions
  - ➢ Found some languages not Regular

- Context Free Grammars
  - ➢ Used to define Context Free Languages (CFLs)
  - ➢ Grammars: Define real languages: Java, C
  - ➢ Recognising languages = Checking syntax
  - ➢ Used in Yacc (and MANY other places)
  - ➢ Grammars come from natural languages

Curtin University of Technology

Background to CFG

- CFG more powerful than regular languages
  - ➢ Can describe features with a recursive structure
  - ➢ Some simple languages such as $\{0^n 1^n \mid n \geq 0\}$ cannot be described by regular expressions

- First used in the study of human languages
  - ➢ Noun, verb, preposition and their respective phrases
  - ➢ Natural recursion *e.g.*, Noun phrases appearing in verb phrases and *vice versa*

Curtin University of Technology

Uses of CFG

- Specification and compilation of programming languages
  - ➢ An important application
- Grammar for a programming language
  - ➢ Reference to learn the language syntax
  - ➢ For design of compilers (constructing a parser from the grammar)

Context-Free Languages

- Collection of languages with context-free grammars
  - ➢ Include regular languages and many others
- Study of CFLs
  - ➢ Formal definition
  - ➢ Properties
  - ➢ Pushdown automata to recognize CFLs
- PDAs provide additional insights into the power of CFGs, and *vice versa*

Curtin University of Technology

Context Free Grammar Terms

- Consider Grammar G_1:
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$
- Consists of a collection of **substitution rules**
  - also called production rules
- Abbreviation within a Grammar:
  - $A \rightarrow 0A1$ and $A \rightarrow B$, written as $A \rightarrow 0A1 \mid B$

Curtin University of Technology

CFG Terms

- Grammar G_1:
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$
- Each rule has a symbol and a string separated by an arrow
  - Symbol is called a **variable** represented by capital letters, *e.g.*, A,B
  - String contains variables and other symbols called **terminals**, *e.g.*, 0,1,#

Curtin University of Technology

CFG Terms

- Grammar G_1:
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$
- Terminals are analogous to the input alphabet
  - represented by lowercase letters, numbers or special symbols
- One variable designated as **start variable**
  - usually in LHS of topmost rule, *e.g.*, A

Generating Strings of the Language

- Write down the start variable
  - usually the variable on the LHS of top rule
- Find a variable that is written down and a rule starting with it
- Replace the variable with RHS of the rule
- Repeat variable replacement until no variable remains

Generating Strings Example

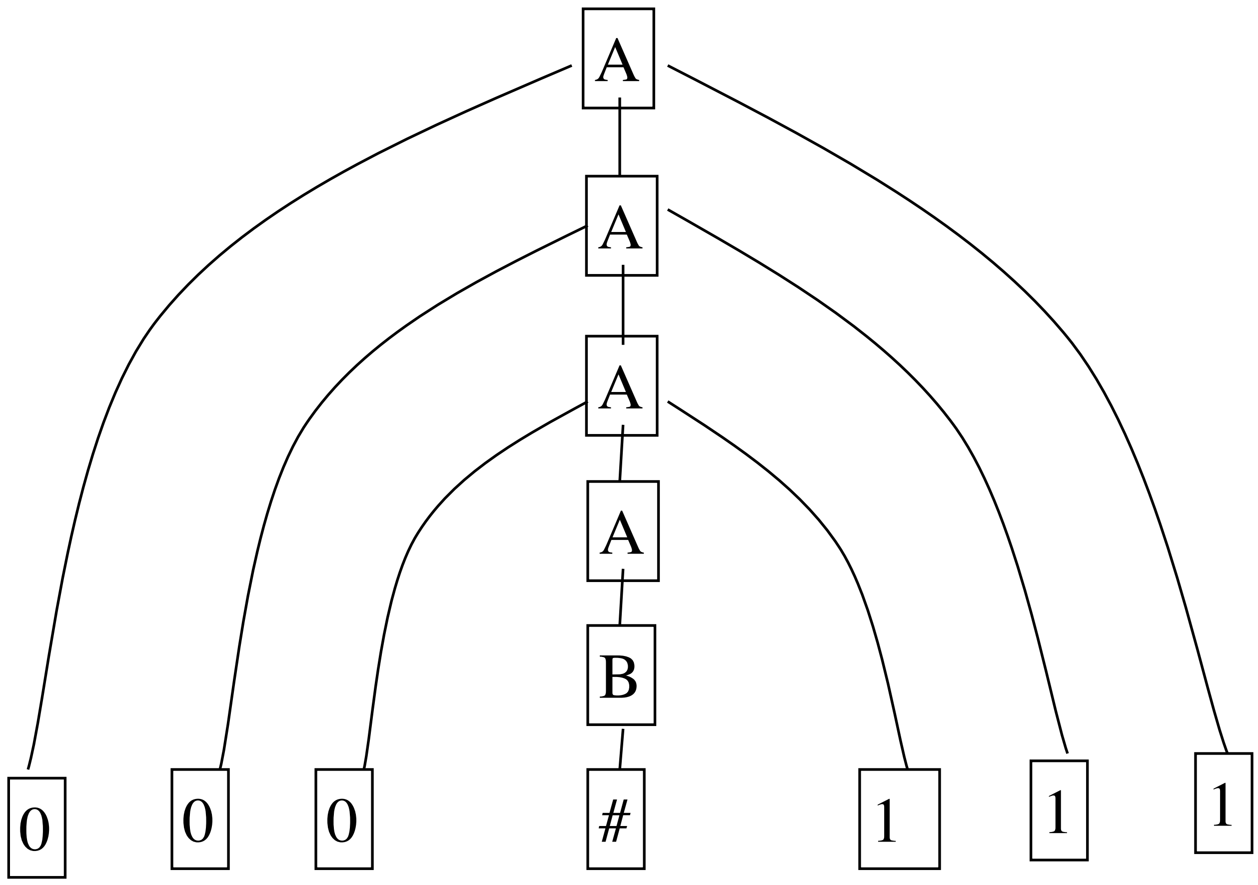
- Grammar G_1:
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$

- G_1 generates the string 000#111

- Derivation: Sequence of substitutions to obtain a string
  - $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$
  - Can also be represented by a parse tree

Curtin University of Technology

Parse Tree



Parse tree for 000#111 in grammar G_1

Language of the Grammar

- Consists of all strings that can be generated using a context-free grammar
  - Called context-free language
  - Written as $L(G_1)$ for the language of CFG G_1
  - $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
- Abbreviation within a Grammar:
  - $A \rightarrow 0A1$ and $A \rightarrow B$, written as $A \rightarrow 0A1 \mid B$

Curtin University of Technology

CFG Example- G_2

- Grammar G_2
  - <SENTENCE> → <NOUN-PHRASE> <VERB-PHRASE>
  - <NOUN-PHRASE> → <CMPLX-NOUN>|<CMPLX-NOUN> <PREP-PHRASE>
  - <VERB-PHRASE> → <CMPLX-VERB> | <CMPLX-VERB> <PREP-PHRASE>
  - <PREP-PHRASE> → <PREP> <CMPLX-NOUN>
  - <CMPLX-NOUN> → <ARTICLE> <NOUN>
  - <CMPLX-VERB> → <VERB> | <VERB> <NOUN-PHRASE>
  - <ARTICLE> → a | the
  - <NOUN> → boy | girl | flower
  - <VERB> → touches | likes | sees
  - <PREP> → with

Curtin University of Technology

Derivation Example

- <SENTENCE> ⇒ <NOUN-PHRASE> <VERB-PHRASE>

  ⇒ <CMPLX-NOUN> <VERB-PHRASE>

  ⇒ <ARTICLE> <NOUN> <VERB-PHRASE>

  ⇒ a <NOUN> <VERB-PHRASE>

  ⇒ a boy <VERB-PHRASE>

  ⇒ a boy <CMPLX-VERB>

  ⇒ a boy <VERB>

  ⇒ a boy sees

Formal definition

- A CFG is a 4-tuple (V, Σ, R, S), where
  - ➢ V is a finite set called the variables,
  - ➢ Σ is a finite set, disjoint from V, called the terminals,
  - ➢ R is a finite set of rules, with each rule comprising an arrow separating a variable and a string of variables and terminals, and
  - ➢ S is the start symbol.

Curtin University of Technology

Formal Definition

- If u,v,w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar,
  - we say that uAv yields uwv, written as $uAv \Rightarrow uwv$.

- $u \Rightarrow^* v$
  - if $u=v$, or
  - there is a sequence $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$, for $k \geq 0$.

- Language of the grammar is $\{w \in \sum^* \mid S \Rightarrow^* w\}$

Curtin University of Technology

Example

- Grammar $G_1 = (V, \Sigma, R, S)$, where
- $V = \{A, B\}$,
- $\Sigma = \{0, 1, \#\}$
- $S = A$
- R consists of
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$

Example 2

- Grammar G_2
- V= {<SENTENCE>, <NOUN-PHRASE>, <VERB-PHRASE>, <CMPLX-NOUN>, <PREP-PHRASE>, <CMPLX-VERB>, <PREP>, <ARTICLE>, <NOUN>, <VERB>, <PREP>}
- Σ = {a, b, c, …, z, " "}
  - " " is the blank symbol
- S= <SENTENCE>
- R consists of rules given earlier

Curtin University of Technology

CONSTRUCTING CFG FOR A LANGUAGE

General Approach

CFG for RL

More Hints

Designing CFGs

- Requires creativity
  - ➢ Even trickier to construct than finite automata because we are more used to writing programs for specific tasks than describing languages with grammars
- Some design techniques
  - ➢ Many CFGs are union of simpler CFGs
  - ➢ If possible, break the CFL into simpler pieces, then construct grammars for each piece
  - ➢ Individual grammars can be easily combined
    - – Put all the rules together
    - – Add a new rule $S \to S_1 | S_2 | \ldots | S_k$, where S_i are the start variables for individual grammars

Curtin University of Technology

Example

- To design a grammar for the language
  - $\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}$
- First construct the grammar
  - $S_1 \rightarrow 0S_1 1 | \varepsilon$ for the language $\{0^n 1^n | n \geq 0\}$
- Then the grammar
  - $S_2 \rightarrow 1S_2 0 | \varepsilon$ for the language $\{1^n 0^n | n \geq 0\}$
- Add the rule
  - $S \rightarrow S_1 | S_2$
- To get the grammar
  - $S \rightarrow S_1 | S_2$
  - $S_1 \rightarrow 0S_1 1 | \varepsilon$
  - $S_2 \rightarrow 1S_2 0 | \varepsilon$

Curtin University of Technology

Note

- CFGs with strings similar to $\{0^n 1^n | n \geq 0\}$
  - ➢ The machine would need to remember the number of 0s to verify that it equals the number of 1s
  - ➢ Use a rule of the form $R \rightarrow uRv$ to generate strings where the portion containing the u's corresponds to the portion containing the v's
  - ➢ See S_1 and S_2 in previous example

Curtin University of Technology

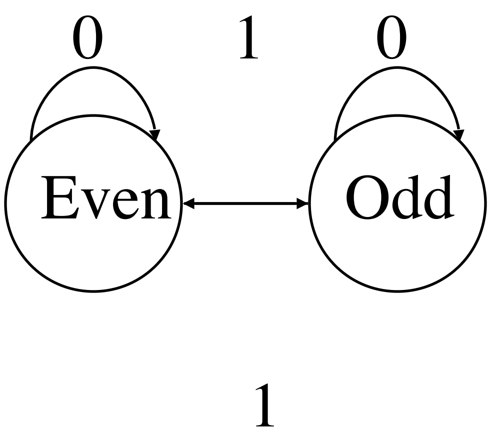
CFG for RL

- Constructing a CFG for a regular language
  - ➢ First construct a DFA for the language
  - ➢ Convert the DFA into an equivalent CFG
    - – See next slide for the method
  - ➢ Verify that the CFG generates the same language that the DFA recognizes

DFA to CFG

- Converting a DFA into an equivalent CFG:
  - Make a variable R_i for each state q_i of the DFA
  - Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_\text{i}, a) = q_\text{j}$ is a transition in the DFA
  - Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA
  - Make R_0 the start variable if q_0 is the start state

Example: Even 1's



The DFA

GRAMMAR

S -> Even
Even -> 0 Even
Even -> 1 Odd
Odd -> 0 Odd
Odd -> 1 Even
Even -> ε

Derivation of 011

S -> Even
-> 0 Even
-> 0 1 Odd
-> 011 Even
-> 011

More Design Techniques

- CFGs for more complex languages
  - Strings may contain certain structures that appear recursively as part of other or the same structures
  - Any time symbol a appears in the example, a parenthesized exp may appear instead
  - Place the variable symbol generating the structure in the location of the rules corresponding to where the structure may recursively appear

- Example:
  - $G_4 = (V, \Sigma, R, \text{<EXPR>})$
  - V is {<EXPR>, <TERM>, <FACTOR>}
  - Σ is {a, +, × , (,)}
  - The rules are
    - <EXPR> → <EXPR>+<TERM> | <TERM>
    - <TERM> → <TERM> × <FACTOR> |<FACTOR>
    - <FACTOR> → (<EXPR>) | a

Curtin University of Technology

AMBIGUITY

Concept
Example
Formal Definition

Curtin University of Technology

Ambiguity

- Sometimes a grammar can generate the same string in several ways
  - ➢ Different parse trees and different meanings
  - ➢ Undesirable for some applications,
    - *e.g.*, Programming languages because a program should have a unique interpretation
- If a grammar generates the same string in several ways, the string is derived ambiguously
  - ➢ Then the grammar is said to be ambiguous

Curtin University of Technology

Example

- Grammar G_5:
  - $\langle EXPR \rangle \to \langle EXPR \rangle + \langle EXPR \rangle | \langle EXPR \rangle \times \langle EXPR \rangle | (\langle EXPR \rangle) | a$
  - G_5 generates the string a+a×a ambiguously
  - $\langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \Rightarrow a + \langle EXPR \rangle \Rightarrow a + \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow a + a \times \langle EXPR \rangle \Rightarrow a + a \times a$
  - $\langle EXPR \rangle \Rightarrow \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow a + \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow a + a \times \langle EXPR \rangle \Rightarrow a + a \times a$

Curtin University of Technology

Example

- Grammar G_4:
  - <EXPR> → <EXPR>+<TERM> | <TERM>
  - <TERM> → <TERM> × <FACTOR> |<FACTOR>
  - <FACTOR> → (<EXPR>) | a
- G_4 generates the same strings as G_5 unambiguously
- <EXPR> ⇒ <EXPR>+<TERM> ⇒ <TERM> +<TERM> ⇒ <FACTOR> +<TERM> ⇒ a +<TERM> ⇒ a + <TERM> × <FACTOR> ⇒ a + <FACTOR> × <FACTOR> ⇒ a + a × <FACTOR> ⇒ a + a × a

Real Languages

- Handling Expressions
  - C favours the G_5 in its definition
    - Yacc has precedence rules to resolve these ambiguities
  - Pascal favoured G_4 in its definition
- Yacc and Ambiguity
  - To resolves ambiguity
    - Uses order of definitions
    - Prefers shift to reduce
      - (Just note it has ambiguity resolution if you have not met Yacc)

Curtin University of Technology

Ambiguous Grammar

- If a string has two different parse trees, the grammar is ambiguous
  - Not two different derivations as derivations may differ only in the order in which variables are replaced
- Leftmost derivation is where at every derivation step, the leftmost remaining variable is replaced
- If a string has two different leftmost derivations, the grammar is ambiguous

Curtin University of Technology

Inherently Ambiguous Languages

- Some CFLs can be generated only by ambiguous grammars
- Example:
  - $\{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$ is an inherently ambiguous language

CHOMSKY NORMAL FORM

Purpose

Definition

Theorem: CFG into CNF

Curtin University of Technology

Purpose of CNF

- CNF: Chomsky Normal Form
  - ➢ Simple Context Free Grammar
    - Yet all CFG can be expressed in its form
    - Used to simplify proofs in decidability

- Note from DFA to CFG
  - ➢ Regular languages of form
    - A -> aB
    - A -> a
    - A -> ε

Curtin University of Technology

Chomsky Normal Form

- Simplified form for CFGs
  - Useful for giving algorithms dealing with CFGs
- Definition: A CFG is in CNF if every rule is of the form
  - $A \rightarrow BC$
  - $A \rightarrow a$

  Where:
  - a is any terminal and A, B, and C are variables.
  - B and C may not be start variables
  - The rule $S \rightarrow \varepsilon$ is allowed where S is the start variable

CNF Theorem

- Any CFL is generated by a CFG in Chomsky normal form.
- Converting any CFG into CNF
  - ➢ Add a new start symbol
  - ➢ Eliminate all rules of the form $A \rightarrow \varepsilon$ and $A \rightarrow B$
  - ➢ Modify the grammar to generate the same language
  - ➢ Convert the remaining rules into the proper form

Curtin University of Technology

Proof of CNF Theorem

- 1. Add a new start symbol S_0 and the rule $S_0 \rightarrow S$, where S was the original start symbol
  - Guarantees that the start symbol does not occur on the RHS of a rule.

- Example: CFG G_6
  - $S \rightarrow ASA|aB$
  - $A \rightarrow B \mid S$
  - $B \rightarrow b \mid \varepsilon$
- *Add a new start symbol*
  - $\boldsymbol{S_0 \rightarrow S}$
  - $S \rightarrow ASA|aB$
  - $A \rightarrow B \mid S$
  - $B \rightarrow b \mid \varepsilon$

Proof of CNF Theorem

- 2. Take care of ε-rules.
  - Remove an ε-rule $A \to \varepsilon$, where A is not a start variable.
  - For each occurrence of an A on the RHS of a rule, add a new rule with that occurrence deleted.
  - If $R \to A$ is a rule, then add $R \to \varepsilon$, unless this rule was previously removed

- *Removing* $B \to \varepsilon$
  - $S_0 \to S$
  - $S \to ASA \mid aB \mid \mathbf{a}$
  - $A \to B \mid S \mid \varepsilon$
  - $B \to b$
- *Removing* $A \to \varepsilon$
  - $S_0 \to S$
  - $S \to ASA \mid aB \mid a \mid \mathbf{SA} \mid \mathbf{AS} \mid \mathbf{S}$
  - $A \to B \mid S$
  - $B \to b$

Curtin University of Technology

Proof of CNF theorem

- 3. Handle all unit rules.
  - Remove a unit rule $A \rightarrow B$
  - For any rule $B \rightarrow u$, add $A \rightarrow u$ unless this rule was previously removed
    - u is a string of variables and terminals
  - *Example: From previous slide*
  - $S_0 \rightarrow S$
  - $S \rightarrow ASA|aB\ |a|SA|AS|S$
  - $A \rightarrow B\ |\ S$
  - $B \rightarrow b$

- Remove $S \rightarrow S$
  - $S_0 \rightarrow S$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow B\ |\ S$
  - $B \rightarrow b$
- Remove $S_0 \rightarrow S$
  - $S_0 \rightarrow \boldsymbol{ASA|aB\ |a|SA|AS}$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow B\ |\ S$
  - $B \rightarrow b$

Curtin University of Technology

Proof of CNF theorem

- 3. Handle all unit rules.
  - Remove a unit rule $A \rightarrow B$
  - For any rule $B \rightarrow u$, add $A \rightarrow u$ unless this rule was previously removed
    - u is a string of variables and terminals
  - Example:
  - $S_0 \rightarrow ASA|aB\ |a|SA|AS$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow B\ |\ S$
  - $B \rightarrow b$
- Remove $A \rightarrow B$
  - $S_0 \rightarrow ASA|aB\ |a|SA|AS$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow S\ |\ \boldsymbol{b}$
  - $B \rightarrow b$
- Remove $A \rightarrow S$
  - $S_0 \rightarrow ASA|aB\ |a|SA|AS$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow b|\boldsymbol{ASA}|\boldsymbol{aB}|\boldsymbol{a}|\boldsymbol{SA}|\boldsymbol{AS}$
  - $B \rightarrow b$

Curtin University of Technology

Proof of CNF Theorem

- 4. Convert all remaining rules into proper form of $A \rightarrow BC$
  - Replace rules like $A \rightarrow u_1 u_1 \ldots u_k$, with $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2 \ldots$
  - Replace any terminal u_i with a new variable U_i and add the rule $U_i \rightarrow u_i$
- *Example: From previous slide*
  - $S_0 \rightarrow ASA|aB\ |a|SA|AS$
  - $S \rightarrow ASA|aB\ |a|SA|AS$
  - $A \rightarrow b|ASA|aB|a|SA|AS$
  - $B \rightarrow b$
- *Using a single variable U and rule $U \rightarrow a$*
  - $S_0 \rightarrow AA_1|UB\ |a|SA|AS$
  - $S \rightarrow AA_1|UB\ |a|SA|AS$
  - $A \rightarrow b|\ AA_1|UB|a|SA|AS$
  - $A_1 \rightarrow SA$
  - $U \rightarrow a$
  - $B \rightarrow b$

Curtin University of Technology

Another Example

- Convert the following CFG into Chomsky normal form, clearly indicating the different steps in the process:
  - $A \rightarrow 0A1$
  - $A \rightarrow B$
  - $B \rightarrow \#$

Curtin University of Technology

Example 2

- 1. Add a new start symbol
  - $S_0 \to A$
  - $A \to 0A1$
  - $A \to B$
  - $B \to \#$
- 2. No ε–rules, so go to next step
- 3. Remove unit rules:
  - Remove $A \to B$
    - $S_0 \to A$
    - $A \to 0A1 \mid \#$
    - $B \to \#$
  - $B \to \#$ *no longer required.*
  - Remove $S_0 \to A$
    - $S_0 \to 0A1 \mid \#$
    - $A \to 0A1 \mid \#$

Curtin University of Technology

Example 2

- 4. Convert to proper form
  - $S_0 \rightarrow CE \mid \#$
  - $A \rightarrow CE \mid \#$
  - $C \rightarrow 0$
  - $D \rightarrow 1$
  - $E \rightarrow AD$

Curtin University of Technology

Equivalence of PDA and CFG

Statement of Theorem

CFG to PDA

PDA to CFG sketched

Curtin
University of Technology

Equivalence of PDA and CFG

- Theorem:
  - A language is context free iff some PDA recognizes it.

- Proof idea:
  - Given any CFL, there is a CFG for it. Convert the CFG to an equivalent PDA.
  - Given a PDA, make a CFG that generates all strings that the PDA accepts.

Curtin University of Technology

CFL to PDA

- A CFL A has a CFG G generating it.
- How to convert G to an equivalent PDA P.
  - P works by accepting its input w, if G generates that input *i.e.*, if there is a derivation for w.
  - Design P to determine whether some series of substitutions using the rules of G can lead from the start variable to w.
  - Difficulty in determining which substitutions to make to generate w.
  - PDA's non-determinism allows it to guess the sequence of correct substitutions

Curtin University of Technology

CFL to PDA

- How the PDA P computes.
  - ➢ P begins by writing the start variable on its stack.
  - ➢ It goes through a series of intermediate strings, making one substitution after another.
  - ➢ Eventually, it may arrive at a string containing only terminal symbols.
  - ➢ If this string matches the input string, the input is accepted.

Informal description of P

1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever.
   a) If the top of the stack is a variable symbol A, non-deterministically select one of the rules for A and substitute A by the RHS of the rule.
   b) If the top of the stack is a terminal symbol a, read the next symbol from input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the non-determinism.
   c) If the top of the stack is the \$ symbol, enter accept state. Doing so accepts input if it has all been read.

Construction of PDA

- Use short hand notation for transition function.
  - Allows writing an entire string on the stack in one step.
  - This action can be simulated using additional states.

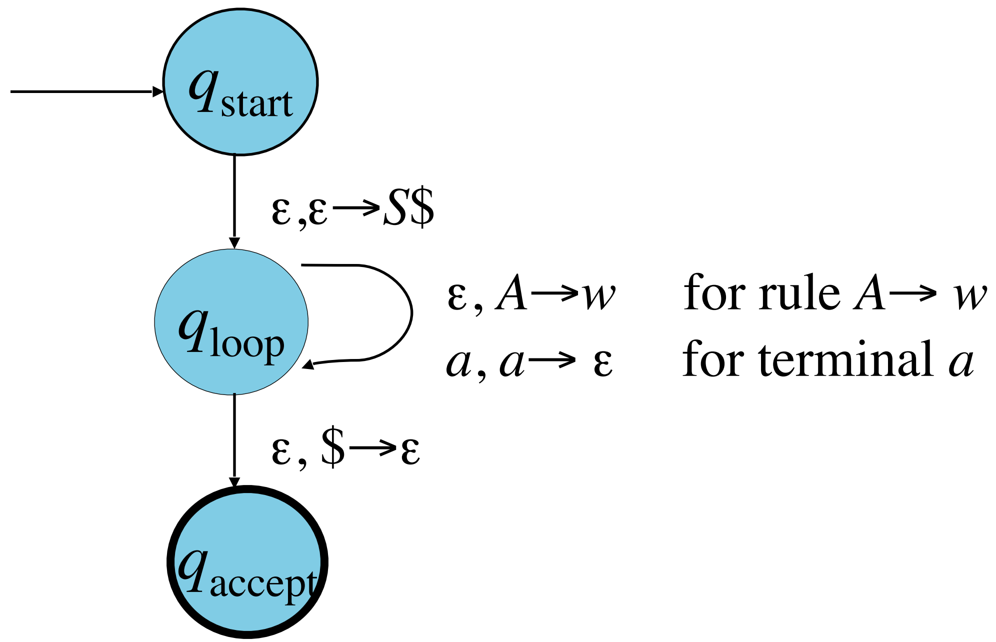
- The start states of P are $Q=\{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$
  - E is the set of states to implement the shorthand.

Construction of PDA

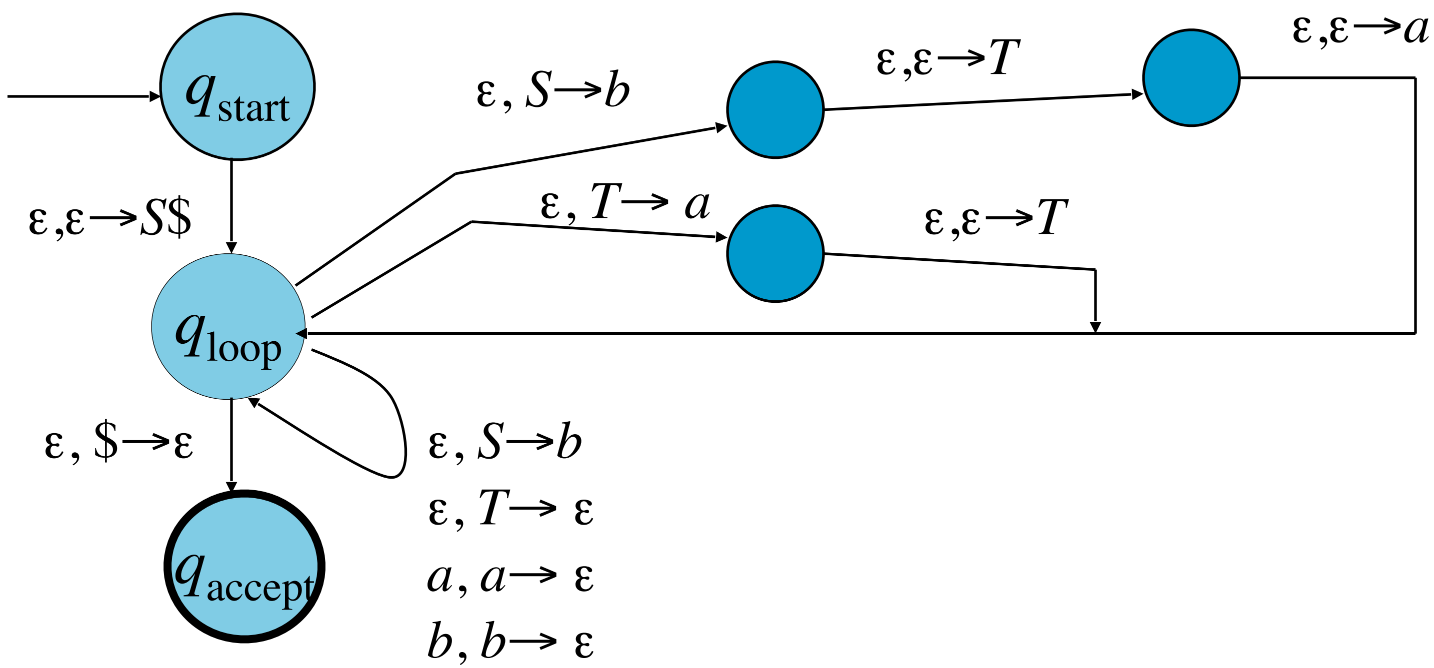
- Transition functions:
- Initialize stack (Step 1 of informal description)
  - $d(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$
- Main loop of step 2:
  - Case (a): variable on top of stack
    - $d(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) | A \rightarrow w \text{ is a rule in R}\}$
  - Case (b): terminal on top of stack
    - $d(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$
  - Case (c): \$ on top of the stack
    - $d(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$

Curtin University of Technology

State Diagram of P



Example

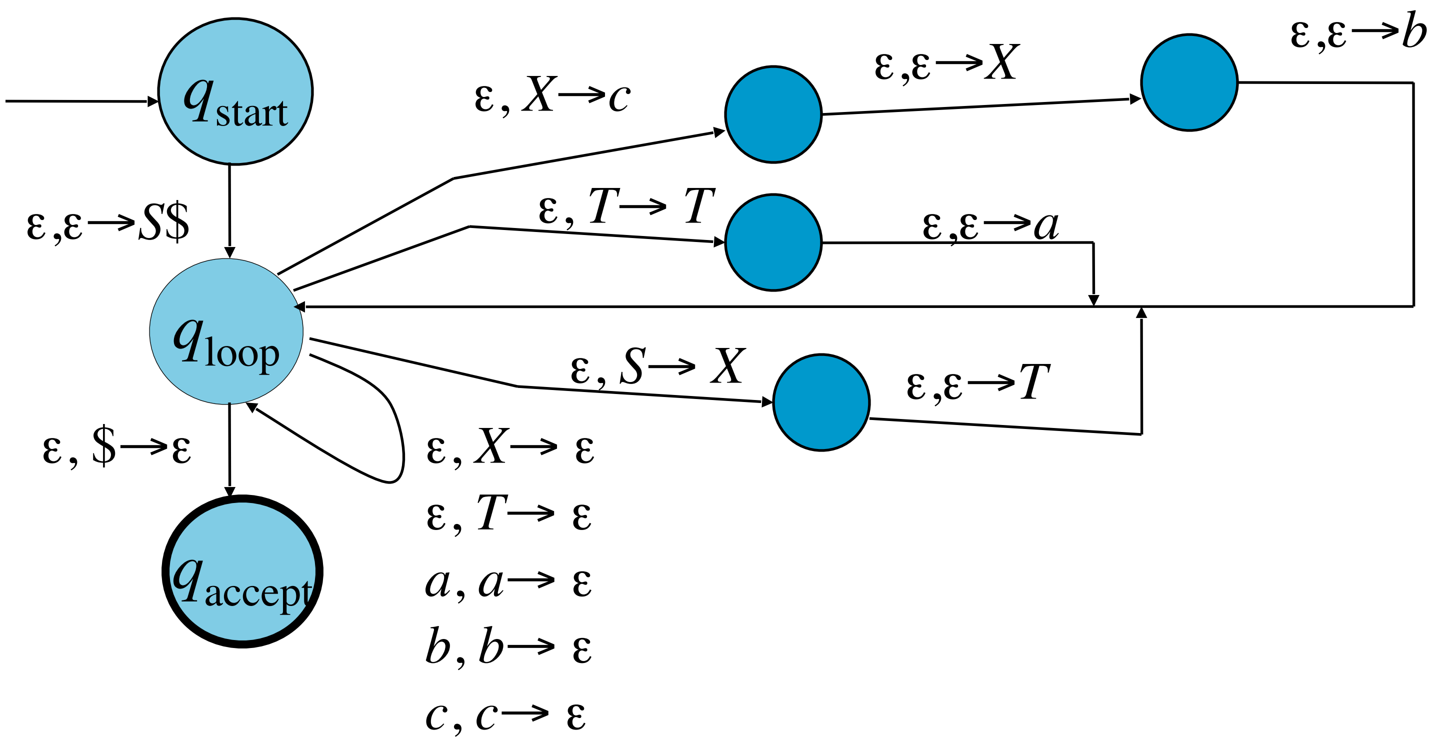


- Construct a PDA from the following CFG G:
  - $S \to \mathrm{a}T\mathrm{b} \mid \mathrm{b}$
  - $T \to T\mathrm{a} \mid \varepsilon$
  - Transition function is shown by the diagram

Example

- Convert the following CFG into an equivalent pushdown automaton, using the construction used for proving that every CFG has an equivalent PDA:
  - $S \rightarrow TX$
  - $T \rightarrow aT \mid e$
  - $X \rightarrow bXc \mid e$
- Give the state diagram and an informal description of the PDA.

Example



q_{start}
$\varepsilon, \varepsilon \to S\$$
q_{loop}
$\varepsilon, \$ \to \varepsilon$
q_{accept}
$\varepsilon, X \to c$
$\varepsilon, \varepsilon \to X$
$\varepsilon, \varepsilon \to b$
$\varepsilon, T \to T$
$\varepsilon, \varepsilon \to a$
$\varepsilon, S \to X$
$\varepsilon, \varepsilon \to T$
$\varepsilon, X \to \varepsilon$
$\varepsilon, T \to \varepsilon$
$a, a \to \varepsilon$
$b, b \to \varepsilon$
$c, c \to \varepsilon$

PDA to CFG

- See text
  - ➢ Very complex
    - – Compiler writing interested only in CFG to PDA
  - ➢ Have to check for each pair (p,q) how it interacts with any other pair (rs): $A_{pq} \rightarrow aA_{rs}b$
  - ➢ Repeat for each triple: $A_{pq} \rightarrow A_{pr}A_{rq}$
  - ➢ Repeat for each p: $A_{pp} \rightarrow \varepsilon$

Curtin University of Technology

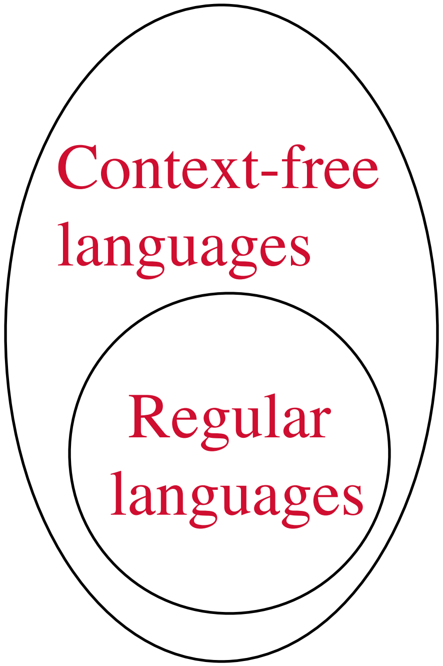
Non-CFL Languages

Relationship of Languages

Pumping lemma

Curtin University of Technology

Relationship of RLs and CFLs



- Every regular language is context-free.

Non-context-free languages

- A technique for proving that some languages are not context free

- Based on a pumping lemma for CFLs

Pumping lemma for CFLs

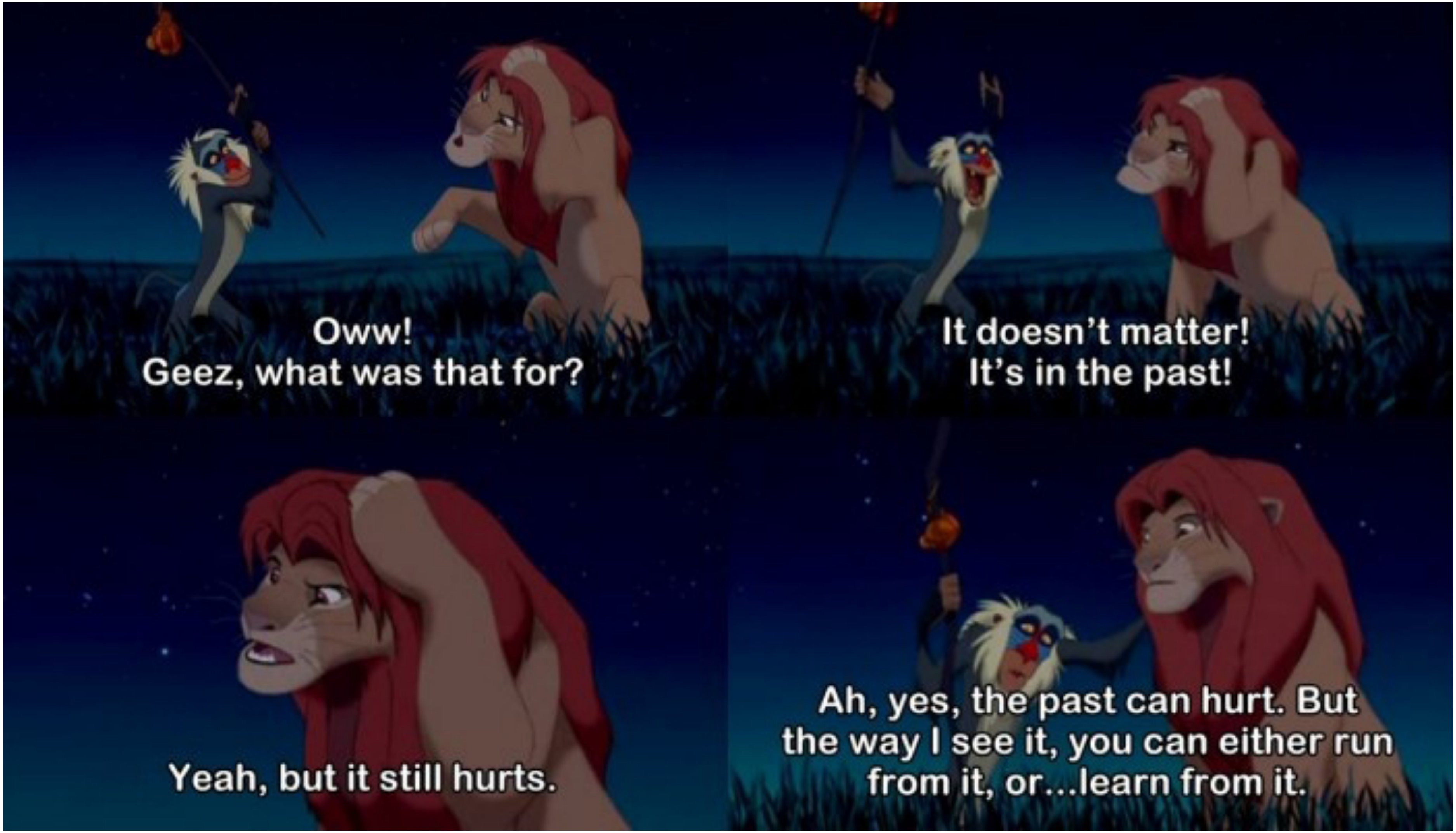
- If A is a context-free language, there is a pumping length p such that if s is any string in A of at least length p, then s may be divided into five pieces, $s = uvxyz$, satisfying the following conditions:
  - for each $i \geq 0$, $uv^i xy^i z \in A$,
  - $|vy| > 0$, and
  - $|vxy| \leq p$.
- The last condition is sometimes useful in proving certain languages to be not CFLs

Curtin
University of Technology

Summary

- Pushdown automaton
  - Formal definition
  - State diagram
  - Examples
- Expressiveness
  - More expressive (powerful) than DFAs
- Non-context free languages
  - Pumping lemma for CFG

This is the end…



But you can't run from the test on September 9.

Curtin University of Technology