Context-Free Grammar and Push-Down Automata

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

November 28, 2008

Outline

Quiz

1. Given 2 DFAs $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $M_2 = (R, \Sigma, \delta_2, r_0, F_2)$, show how to construct a DFA M_3 that recognizes language $A_1 \cap A_2$ where A_i is a language recognized by M_i for each $i = 1, 2$.
2. Show that if A_1 and A_2 are regular languages, $A_1 - A_2$ is also regular.

Regular languages

These sets of languages are equal:

- a set of languages recognized by deterministic finite automata,
- a set of languages recognized by nondeterministic finite automata, and
- a set of languages described by regular expressions

Regular languages

These sets of languages are equal:

- a set of languages recognized by deterministic finite automata,
- a set of languages recognized by nondeterministic finite automata, and
- a set of languages described by regular expressions

They are **regular languages**.

How to prove the equivalence?

To prove that, for two sets A and B, $A = B$, we can

How to prove the equivalence?

To prove that, for two sets A and B, $A = B$, we can

- First show that $A \subseteq B$,

How to prove the equivalence?

To prove that, for two sets A and B, $A = B$, we can

- First show that $A \subseteq B$, and then
- show that $B \subseteq A$.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language. Easy.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language. Easy.
- Given an NFA N, construct a DFA M recognizing the same language.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language. Easy.
- Given an NFA N, construct a DFA M recognizing the same language. Not so hard.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language. Easy.
- Given an NFA N, construct a DFA M recognizing the same language. Not so hard.
  - Let M simulates N by keeping the set of states that copies of N are in.

DFAs & NFAs

- Given a DFA M, construct an NFA N recognizing the same language. Easy.
- Given an NFA N, construct a DFA M recognizing the same language. Not so hard.
  - Let M simulates N by keeping the set of states that copies of N are in.

Thus, the sets of languages recognized by DFAs and NFAs are equal.

(D/N)FAs and regular expressions

An example

Grammar G_1

$$
\begin{array}{lcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with A

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1)

An example

Grammar G_1

$$
\begin{array}{rcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1)

An example

Grammar G_1

$$\begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \\ B &\rightarrow \# \end{aligned}$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2)

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2) $\Rightarrow 00\#11$ (rule 3).

An example

Grammar G_1

$$\begin{array}{lcl} A & \rightarrow & 0A1 \\ A & \rightarrow & B \\ B & \rightarrow & \# \end{array}$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2) $\Rightarrow 00\#11$ (rule 3).

This sequence of substitution is called a **derivation**.

A grammar

From previous example, you may notice that the grammar has

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow),

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow), and
- terminals (other symbols).

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow), and
- terminals (other symbols).

To obtain a derivation, we also need a start variable.

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow), and
- terminals (other symbols).

To obtain a derivation, we also need a start variable. (If not specified otherwise, it is the left-hand side of the top rule.)

From the start variable

- The grammar G_1 **generates** the string 000#111.

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.

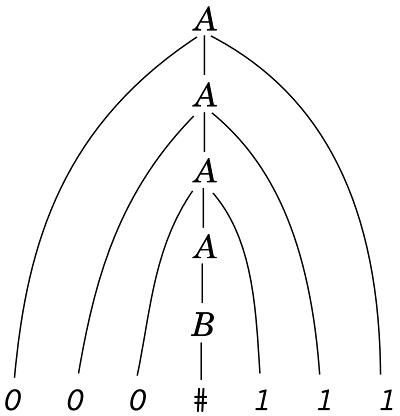
From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.
  - Find a variable in the string and a rule that starts with that variable. Replace the variable with the right-hand side of the rule.

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.
  - Find a variable in the string and a rule that starts with that variable. Replace the variable with the right-hand side of the rule.
  - Repeat.

A parse tree



Language of the grammar

- A grammar **describes** a language by generating each string of the language.

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.
- $L(G_1) =$

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.
- $L(G_1) = \{0^n\#1^n | n \geq 0\}$

A context-free language

A language described by some context-free grammar is called a context-free language.

More example

Grammar G_2

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
NP &\rightarrow CN | CN\ PP \\
VP &\rightarrow CV | CV\ PP \\
PP &\rightarrow PREP\ CN \\
CN &\rightarrow ART\ N \\
CV &\rightarrow V | V\ NP \\
ART &\rightarrow \texttt{a} | \texttt{the} \\
N &\rightarrow \texttt{boy} | \texttt{girl} | \texttt{flower} \\
V &\rightarrow \texttt{touches} | \texttt{likes} | \texttt{sees} \\
PREP &\rightarrow \texttt{with}
\end{aligned}
$$

Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`

Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`
  - `the boy sees a flower`

Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`
  - `the boy sees a flower`
  - `a girl with a flower likes the boy`

Derivation

- Show the derivation of string "`a boy sees`".

Derivation

- Show the derivation of string "`a boy sees`".
- Try to generate more strings from G_2 and find their parse trees.

Definition [context-free grammar]

Definition

A **context-free grammar** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the **start variable**.

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv,

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \longrightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v,

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \overset{*}{\Rightarrow} v$,

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \stackrel{*}{\Rightarrow} v$,
  - if $u = v$, or

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \to w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \stackrel{*}{\Rightarrow} v$,
  - if $u = v$, or
  - if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

Example: G_3

$G_3 = (\{S\}, \{a, b\}, R, S)$, where R is

$$S \rightarrow aSb|SS|\varepsilon.$$

Practice

Find a CFG that describes the following language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$$

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,
- the rules are

$$EXPR \rightarrow EXPR + EXPR \mid EXPR \times EXPR \mid (EXPR) \mid a$$

Generate some string from G_4'.

Ambiguity

Find a parse tree for $a + a \times a$ in grammar G_4'.

Example: G_4

$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,

Example: G_4

$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,

Example: G_4

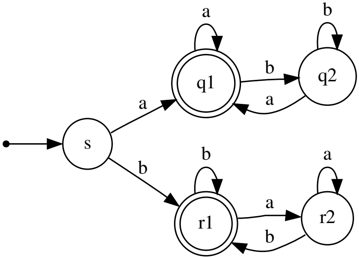
$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,
- the rules are

$$
\begin{aligned}
EXPR &\rightarrow EXPR + TERM | TERM \\
TERM &\rightarrow TERM \times FACTOR | FACTOR \\
FACTOR &\rightarrow (EXPR) | a
\end{aligned}
$$

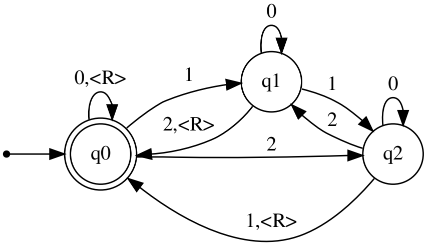
CFGs and regular languages (1)

Can you find a context-free grammar that describes the language recognized by the following DFA?



CFGs and regular languages (2)

Can you find a context-free grammar that describes the language recognized by the following DFA?



Again, think about a “mechanical” procedure for constructing a CFG.

CFGs and regular languages (3)

Any general procedure?

Simpler forms

- Since we know that DFAs and NFAs are equivalent,

Simpler forms

- Since we know that DFAs and NFAs are equivalent, we can pick one that allow us to prove the property that we want.

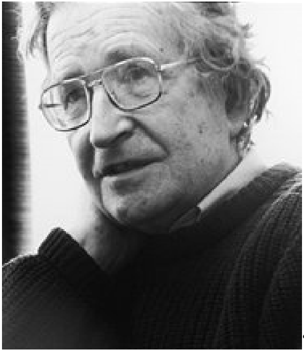
Simpler forms

- Since we know that DFAs and NFAs are equivalent, we can pick one that allow us to prove the property that we want.
- Again, CFGs is quite general and sometimes we want them to be in a simpler form.

Simpler forms

- Since we know that DFAs and NFAs are equivalent, we can pick one that allow us to prove the property that we want.
- Again, CFGs is quite general and sometimes we want them to be in a simpler form.
- One of the forms is called Chomsky normal form.

Noam Chomsky

[a]

Avram Noam Chomsky is an American linguist, philosopher, cognitive scientist, political activist, author, and lecturer. [from wikipedia]

[a] From wikipedia. URL:
`http://en.wikipedia.org/wiki/Image:Noam_chomsky_cropped.jpg`

Definition [Chomsky normal form]

A context-free grammar is in **Chomsky normal form** is every rule is of the form

$$A \longrightarrow BC$$

$$A \longrightarrow a$$

where a is any terminal and A, B, and C are any variables,

Definition [Chomsky normal form]

A context-free grammar is in **Chomsky normal form** is every rule is of the form

$$A \longrightarrow BC$$

$$A \longrightarrow a$$

where a is any terminal and A, B, and C are any variables, except that B and C cannot be the start variable.

Definition [Chomsky normal form]

A context-free grammar is in **Chomsky normal form** is every rule is of the form

$$A \longrightarrow BC$$

$$A \longrightarrow a$$

where a is any terminal and A, B, and C are any variables, except that B and C cannot be the start variable. We also permit the rule $S \longrightarrow \varepsilon$, where S is the start variable.

Theorem 1

Any context-free grammar is generated by a context-free grammar in Chomsky normal form.

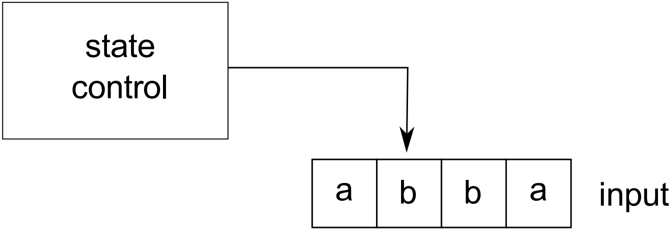
Pushdown automata

- NFAs power-up

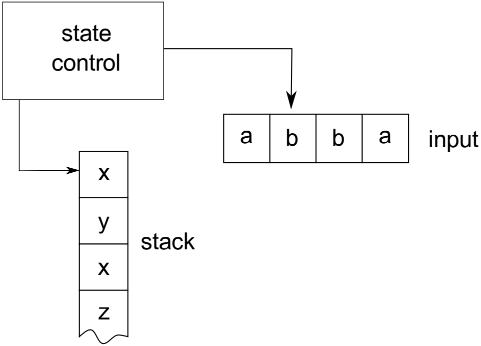
Pushdown automata

- NFAs power-up
- Think of them as NFAs with extra memory, called **stack**.

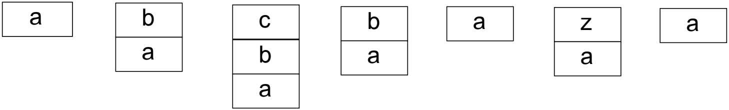
NFAs



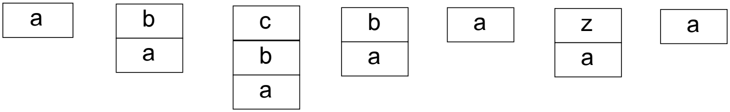
PDas



Stacks

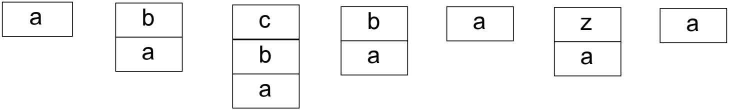


Stacks



A stack is an infinite memory but you can only access the topmost element.

Stacks



A stack is an infinite memory but you can only access the topmost element.
You can pop (put something on top) and push (remove the topmost).

Informally

Can you find an NFA with a stack that recognizes $\{0^n 1^n | n \geq 0\}$?

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.
- How does a PDA move?

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.
- How does a PDA move?
    - It reads some input (can be ε).

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.
- How does a PDA move?
  - It reads some input (can be ε).
  - It reads the top of the stack (can be ε as well).

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.
- How does a PDA move?
  - It reads some input (can be ε).
  - It reads the top of the stack (can be ε as well).
  - It changes the state and writes something to the top of the stack.
- Thus, the transition function accepts (q, x, s) where q is a state, x is an input symbol, and s is the top of the stack.

Transition function with stack (1)

- A stack keeps some data. Let Γ be a stack alphabet.
- How does a PDA move?
  - It reads some input (can be ε).
  - It reads the top of the stack (can be ε as well).
  - It changes the state and writes something to the top of the stack.
- Thus, the transition function accepts (q, x, s) where q is a state, x is an input symbol, and s is the top of the stack.
- The transition function returns a set of pairs (q', s') where q' is a new state and s' is the stack symbol written to the stack.

Transition function with stack (2)

- Transition function δ:
  - Domain: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$
  - Range: $\mathcal{P}(Q \times \Gamma_\varepsilon)$

Definition [pushdown automaton]

A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.