CS21
Decidability and Tractability

Lecture 6

January 16, 2015

Outline

- Context-Free Grammars and Languages
  - parse trees
  - ambiguity
  - normal form
- equivalence of NPDAs and CFGs
- non context-free languages

Context-Free Grammars

start symbol

terminal symbols

A → 0A1

A → B

B → #

non-terminal symbols

production

CFG example

- Arithmetic expressions over {+,*,(,),a}
  - (a + a) * a
  - a * a + a + a + a + a
- A CFG generating this language:

  <expr> → <expr> * <expr>

  <expr> → <expr> + <expr>

  <expr> → (<expr>) | a

CFG example

<expr> → <expr> * <expr>
<expr> → <expr> + <expr>
<expr> → (<expr>) | a

- A derivation of the string: a+a*a

  <expr> ⇒ <expr> * <expr>

  ⇒ <expr> + <expr> * <expr>

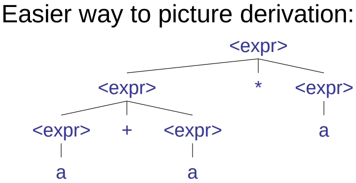
  ⇒ a + <expr> * <expr>

  ⇒ a + a * <expr>

  ⇒ a + a * a

Parse Trees

- Easier way to picture derivation: parse tree



- grammar encodes grouping information; this is captured in the parse tree.

CFGs and parse trees

<expr> → <expr> * <expr>
<expr> → <expr> + <expr>
<expr> → (<expr>) | a

- Is this a good grammar for arithmetic expressions?
  – can group wrong way (+ precedence over *)
  – can also group correct way (ambiguous)

Solution to first problem

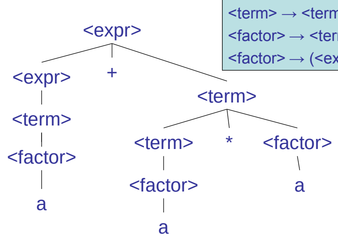
<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → <term> * <factor>
<factor> → (<expr>) | a

- forces correct precedence in parse tree grouping
  – within parentheses, * cannot occur as ancestor of + in the parse tree.

Parse Trees

- parse tree for a + a * a in new grammar:

<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → <term> * <factor>
<factor> → (<expr>) | a



Ambiguity

- Second problem: ambiguous grammar
- Definitions:
  – a string is derived ambiguously if it has two different parse trees
  – a grammar is ambiguous if its language contains an ambiguously derived string
- ambiguity sometimes undesirable
- some CFLS are inherently ambiguous

Ambiguity

- Definition in terms of derivations (rather than parse trees):
  – order in which we replace terminals in shouldn't matter (often several orders possible)
  – define leftmost derivation to be one in which the leftmost non-terminal is always the one replaced
  – a string is ambiguously derived if it has 2 leftmost derivations

Chomsky Normal Form

- Useful to deal only with CFGs in a simple normal form
- Most common: Chomsky Normal Form (CNF)
- Definition: every production has form

  $A \to BC$ or $S \to \varepsilon$ or $A \to a$

  where A, B, C are any non-terminals (and B, C are not S) and a is any terminal.

Chomsky Normal Form

Theorem: Every CFL is generated by a CFG in Chomsky Normal Form.

Proof: Transform any CFG into an equivalent CFG in CNF. Four steps:
- add a new start symbol
- remove "ε-productions" $A \to \varepsilon$
- eliminate "unit productions" $A \to B$
- convert remaining rules into proper form

Chomsky Normal Form

- add a new start symbol
  - add production $S_0 \to S$
- remove "ε-productions" $A \to \varepsilon$
  - for each production with A on rhs, add production with A's removed: e.g. for each rule $R \to uAv$, add $R \to uv$
- eliminate "unit productions" $A \to B$
  - for each production with B on lhs: $B \to u$, add rule $A \to u$

Chomsky Normal Form

- convert remaining rules into proper form
  - replace production of form:

$$A \to u_1U_2u_3\ldots u_k$$

with:

$A \to U_1A_1$ $\quad U_1 \to u_1$

$A_1 \to U_2A_2$ $\quad$ U_2 already a non-terminal

$A_2 \to U_3A_3$ $\quad U_3 \to u_3$

:

$A_{k-2} \to U_{k-1}U_k$ $\quad U_{k-1} \to u_{k-1}$ $\quad U_k \to u_k$

Some facts about CFLs

- CFLs are closed under
  - union (proof?)
  - concatenation (proof?)
  - star (proof?)
- Every regular language is a CFL
  - proof?

NPDA, CFG equivalence

Theorem: a language L is recognized by a NPDA iff L is described by a CFG.

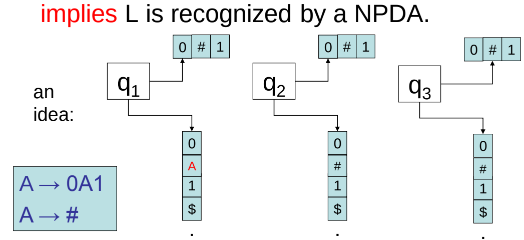
Must prove *two* directions:

($\Rightarrow$) L is recognized by a NPDA implies L is described by a CFG.

($\Leftarrow$) L is described by a CFG implies L is recognized by a NPDA.

NPDA, CFG equivalence

Proof of ($\Leftarrow$): L is described by a CFG implies L is recognized by a NPDA.



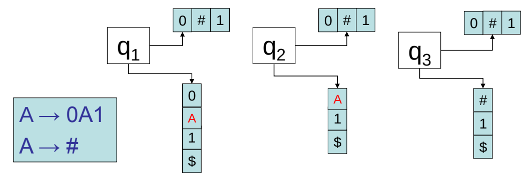
NPDA, CFG equivalence

1. we'd like to non-deterministically guess the derivation, forming it on the stack
2. then scan the input, popping matching symbol off the stack at each step
3. accept if we get to the bottom of the stack at the end of the input.

what is wrong with this approach?

NPDA, CFG equivalence

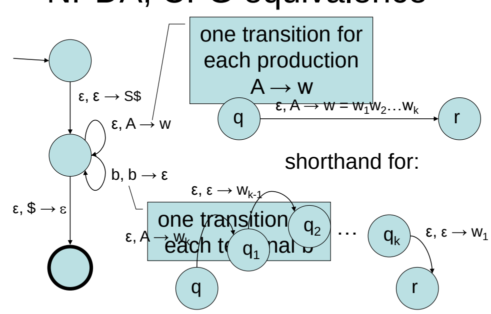
– only have access to top of stack
– combine steps 1 and 2:
  - allow to match stack terminals with tape *during* the process of producing the derivation on the stack



NPDA, CFG equivalence

- informal description of construction:
  – place $ and start symbol S on the stack
  – repeat:
    - if the top of the stack is a non-terminal A, pick a production with A on the lhs and substitute the rhs for A on the stack
    - if the top of the stack is a terminal b, read b from the tape, and pop b from the stack.
    - if the top of the stack is $, enter the accept state.

NPDA, CFG equivalence



NPDA, CFG equivalence

Proof of (⇒): L is recognized by a NPDA implies L is described by a CFG.

– harder direction
– first step: convert NPDA into "normal form":
  - single accept state
  - empties stack before accepting
  - each transition *either* pushes *or* pops a symbol

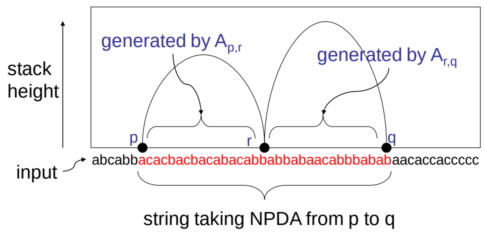
NPDA, CFG equivalence

– **main idea**: non-terminal $A_{p,q}$ generates exactly the strings that take the NPDA from state p (w/ empty stack) to state q (w/ empty stack)

– then $A_{start, accept}$ generates all of the strings in the language recognized by the NPDA.

NPDA, CFG equivalence

- Two possibilities to get from state p to q:



NPDA, CFG equivalence

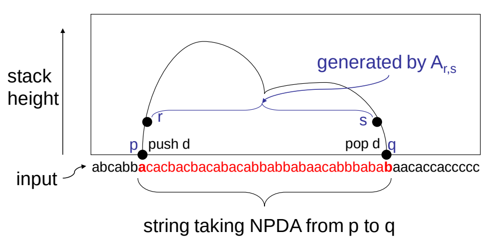
- NPDA P = (Q, Σ, Γ, δ, start, {accept})
- CFG G:
  - non-terminals V = $\{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start, accept}$
  - productions:

    for every p, r, q ∈ Q, add the rule

    $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

NPDA, CFG equivalence

- Two possibilities to get from state p to q:



NPDA, CFG equivalence

- NPDA P = (Q, Σ, Γ, δ, s
- CFG G:
  - non-terminals V = {A
  - start variable $A_{start, ac}$
  - productions:

    for every p, r, s, q ∈ Q, d ∈ Γ, and a, b ∈ (Σ ∪ {ε})

    if (r, d) ∈ δ(p, a, ε), and

    (q, ε) ∈ δ(s, b, d), add the rule

    $A_{p,q} \rightarrow aA_{r,s}b$

from state p, read a, push d, move to state r

from state s, read b, pop d, move to state q

NPDA, CFG equivalence

- NPDA P = (Q, Σ, Γ, δ, start, {accept})
- CFG G:
  - non-terminals V = $\{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start, accept}$
  - productions:

    for every p ∈ Q, add the rule

    $A_{p,p} \rightarrow \varepsilon$