CS21
Decidability and Tractability

Lecture 7

January 21, 2015

Outline

- equivalence of NPDAs and CFGs (finishing up)
- non context-free languages

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 (done last lecture)

NPDA, CFG equivalence

Proof of ($\Rightarrow$): 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

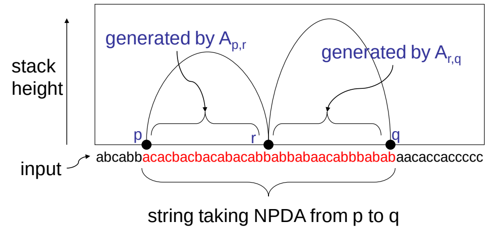
NPDA, CFG equivalence

– **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

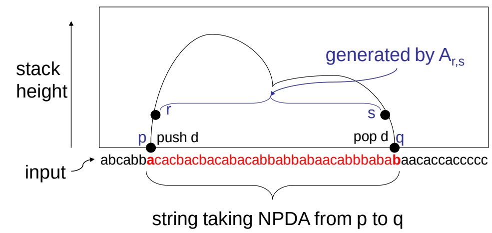
- NPDA P = (Q, Σ, Γ, δ, start, {accept})
- CFG G:
  - non-terminals V = {$A_{p,q}$: p, q ∈ Q}
  - start variable $A_{start,\ accept}$
  - productions:

    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

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

- CFG G:
  - non-terminals V = {A

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

  - 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

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

NPDA, CFG equivalence

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

    for every p ∈ Q, add the rule

    $A_{p,p} \rightarrow ε$

NPDA, CFG equivalence

- two claims to verify correctness:

1. if $A_{p,q}$ generates string x, then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x, then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
   - induction on length of derivation of x.
   - base case: 1 step derivation. must have only terminals on rhs. In G, must be production of form $A_{p,p} \rightarrow ε$.

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x, then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
   - assume true for derivations of length at most k, prove for length k+1.
   - verify case: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \rightarrow^k x = yz$
   - verify case: $A_{p,q} \rightarrow aA_{r,s}b \rightarrow^k x = ayb$

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
   - induction on # of steps in P's computation
   - base case: 0 steps. starts and ends at same state p. only has time to read empty string ε.
   - G contains $A_{p,p} \rightarrow \varepsilon$.

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
   - induction step. assume true for computations of length at most k, prove for length k+1.
   - if stack becomes empty sometime in the middle of the computation (at state r)
     - y is read going from state p to r ($A_{p,r} \rightarrow^* y$)
     - z is read going from state r to q ($A_{r,q} \rightarrow^* z$)
     - conclude: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \rightarrow^* yz = x$

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
   - if stack becomes empty only at beginning and end of computation.
     - first step: state p to r, read a, push d
     - go from state r to s, read string y ($A_{r,s} \rightarrow^* y$)
     - last step: state s to q, read b, pop d
     - conclude: $A_{p,q} \rightarrow aA_{r,s}b \rightarrow^* ayb = x$

Pumping Lemma for CFLs

<u>CFL Pumping Lemma</u>: Let L be a CFL. There exists an integer p ("pumping length") for which every $w \in L$ with $|w| \geq p$ can be written as

$w = uvxyz$ such that

1. for every $i \geq 0$, $uv^ixy^iz \in L$, and
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

CFL Pumping Lemma Example

<u>Theorem</u>: the following language is not context-free:

$$L = \{a^nb^nc^n : n \geq 0\}.$$

- Proof:
  - let p be the pumping length for L
  - choose $w = a^pb^pc^p$

    w = aaaa…abbbb…bcccc…c
  - w = uvxyz, with $|vy| > 0$ and $|vxy| \leq p$.

CFL Pumping Lemma Example

– possibilities:

$$w = \underbrace{aaaa\ldots}_{u}\underbrace{aaa}_{v}\underbrace{bbb\ldots}_{x}\underbrace{bb}_{y}\underbrace{cccc\ldots c}_{z}$$

(if v, y each contain only one type of symbol, then pumping on them produces a string not in the language)

CFL Pumping Lemma Example

– possibilities:

$$w = \underbrace{aaaa\ldots}_{u}\underbrace{ab}_{v}\underbrace{bbb\ldots}_{x}\underbrace{bc}_{y}\underbrace{cccc\ldots c}_{z}$$

(if v or y contain more than one type of symbol, then pumping on them might produce a string with equal numbers of a's, b's, and c's – if vy contains equal numbers of a's, b's, and c's. But they will be out of order.)

CFL Pumping Lemma Example

Theorem: the following language is not context-free:

$$L = \{xx : x \in \{0,1\}^*\}.$$

- Proof:
  - let p be the pumping length for L
  - try $w = 0^p10^p1$
  - can this be pumped?

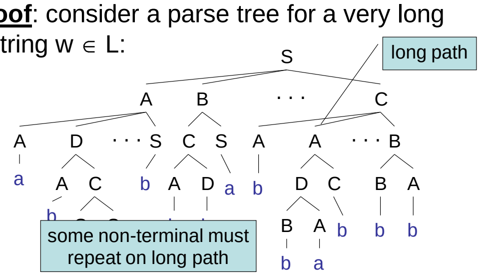
CFL Pumping Lemma Example

$$L = \{xx : x \in \{0,1\}^*\}.$$

- try $w = 0^{2p}1^{2p}0^{2p}1^{2p}$
- $w = uvxyz$, with $|vy| > 0$ and $|vxy| \le p$.
- case: vxy in first half.
  - then $uv^2xy^2z = 0??\ldots?1??\ldots?$
- case: vxy in second half.
  - then $uv^2xy^2z = ??\ldots?0??\ldots?1$
- case: vxy straddles midpoint
  - then $uv^0xy^0z = uxz = 0^{2p}1^i0^j1^{2p}$ with $i \ne 2p$ or $j \ne 2p$

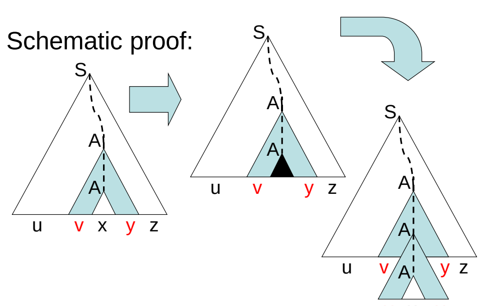
CFL Pumping Lemma

Proof: consider a parse tree for a very long string $w \in L$:



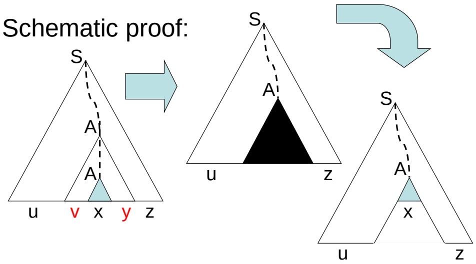
CFL Pumping Lemma

- Schematic proof:



CFL Pumping Lemma

- Schematic proof:



CFL Pumping Lemma

– how large should pumping length p be?

– need to ensure other conditions:

$|vy| > 0$ $|vxy| \leq p$

– b = max # symbols on rhs of any production (assume $b \geq 2$)

– if parse tree has height $\leq h$, then string generated has length $\leq b^h$ (so length $> b^h$ implies height $> h$)

CFL Pumping Lemma

– let m be the # of nonterminals in the grammar

– to ensure path of length at least m+2, require

$|w| \geq \mathbf{p} = b^{m+2}$

– since $|w| > b^{m+1}$, any parse tree for w has height > m+1

– let T be the smallest parse tree for w

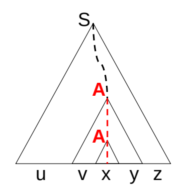
– longest root-leaf path must consist of ≥ m+1 non-terminals and 1 terminal.

CFL Pumping Lemma

– must be a repeated non-terminal **A** on long path

– select a repetition among the lowest m+1 non-terminals on path.

– pictures show that for every $i \geq 0$, $uv^ixy^iz \in L$



– is $|vy| > 0$?

- smallest parse tree T ensures

– is $|vxy| \leq p$?

- red path has length ≤ m+2, so $\leq b^{m+2}$ = p leaves

Deterministic PDA

- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
  – $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \wp(Q \times (\Gamma \cup \{\varepsilon\}))$ is a function called the transition function
- A deterministic PDA has only one option at every step:
  – for every state $q \in Q$, $a \in \Sigma$, and $t \in \Gamma$, exactly 1 element in $\delta(q, a, t)$, or
  – exactly 1 element in $\delta(q, \varepsilon, t)$, and $\delta(q, a, t)$ empty for all $a \in \Sigma$

Deterministic PDA

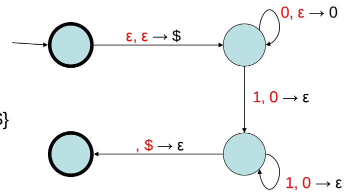
- A technical detail:

  we will give our deterministic machine the ability to detect end of input string
  – add special symbol to alphabet
  – require input tape to contain x
- language recognized by a deterministic PDA is called a deterministic CFL (DCFL)

Example deterministic PDA

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, \$\}$



$L = \{0^n1^n : n \geq 0\}$

(unpictured transitions go to a "reject" state and stay there)