CS21
Decidability and Tractability

Lecture 6

January 17, 2014

Outline

- equivalence of NPDAs and CFGs
- non context-free languages
- deterministic PDAs

Context-Free Grammars

start symbol

terminal symbols

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

non-terminal symbols

production

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.

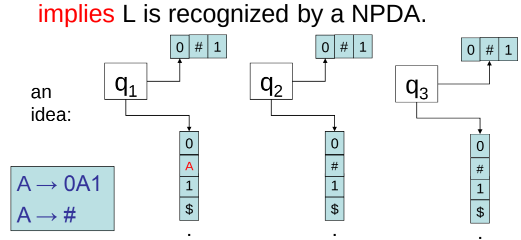
Must prove *two* directions:

($\Rightarrow$) 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.



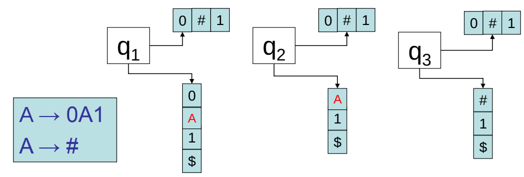
NPDA, CFG equivalence

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

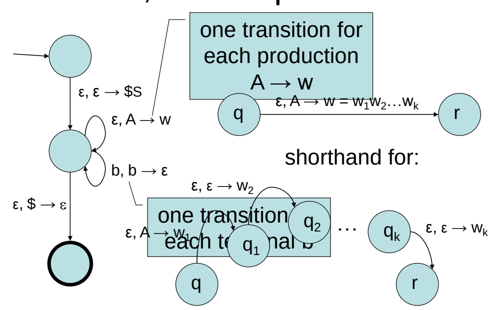
– only have access to top of stack
– combine steps 1 and 2:
  • allow to match stack terminals with tape *during* the process of producing the derivation on the stack



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

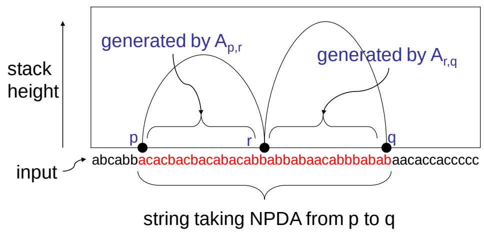
NPDA, CFG equivalence

– **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

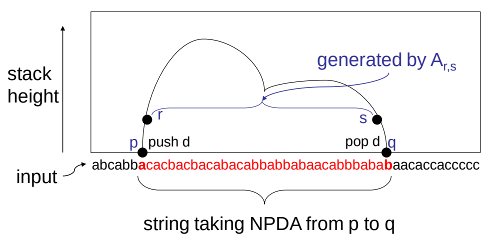
- NPDA P = (Q, Σ, Γ, δ, start, {accept})
- CFG G:
  - non-terminals V = $\{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start,\ accept}$
  - productions:

    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, start, {accept})
- CFG G:
  - non-terminals V = $\{A_{p,q} : p, q \in Q\}$
  - start variable $A_{start,\ accept}$
  - 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

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

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

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

CFL Pumping Lemma: 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

Theorem: the following language is not context-free:

$L = \{a^n b^n c^n : n \geq 0\}$.

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

    $w = aaaa\ldots abbbb\ldots bcccc\ldots c$
  - $w = uvxyz$, 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 $w = 0^p 1 0^p 1$
  - 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 $|vxy| \leq p$.
- 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 $i \neq 2p$ or $j \neq 2p$