Context-Free Grammars and Pushdown Automata

CS114 Lab 8
March 23, 2018
Kenneth Lai

Review: Regular languages

- A language is regular iff:
  - It is recognized by a finite state automaton
  - It is generated by a regular expression

Natural languages are not regular

- Why not?

Context-free languages

- More powerful than regular languages
- Used in specification of natural and programming languages

Context-free languages

- A language is context-free iff:
  - It is recognized by a pushdown automaton
  - It is generated by a context-free grammar

Phrase structure grammars = context-free grammars

- G = (T, N, S, R)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
- A grammar G generates a language L.

Example

- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$

A phrase structure grammar

- S → NP VP
- VP → V NP
- VP → V NP PP
- NP → NP PP
- NP → N
- NP → e
- NP → N N
- PP → P NP

- N → cats
- N → claws
- N → people
- N → scratch
- V → scratch
- P → with

- By convention, S is the start symbol, but in the PTB, we have an extra node at the top (ROOT, TOP)

Chomsky Normal Form

- Every rule is of the form
  - $S \rightarrow \varepsilon$
  - $A \rightarrow BC$
  - $A \rightarrow a$
- Where S is the start symbol, A is a nonterminal, B and C are nonterminals (except for S), and a is a terminal

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove e-rules
- Remove unit rules
- Break up rules with more than 3 things on the right hand side
- Replace terminals with nonterminals and add new rules as needed

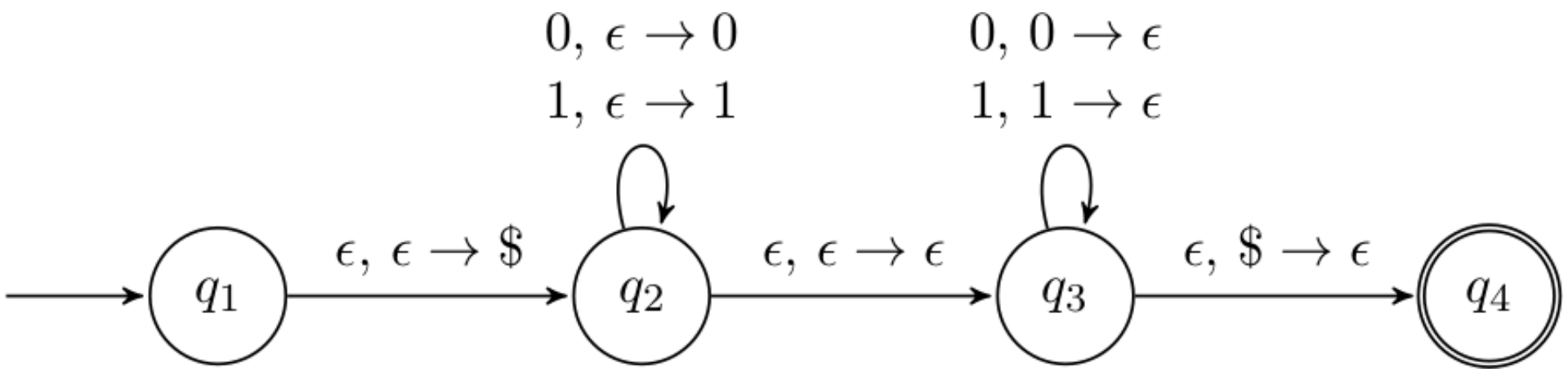
Pushdown automata

- Finite state automata with stacks
  - Infinite memory that can only be used last in, first out

Pushdown automata

- A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  - Q is the set of states
  - Σ is the input alphabet
  - Γ is the stack alphabet
  - δ is the transition function
  - q_0 is the start state
  - F is the set of accept states

Example


$0,\ \epsilon \to 0$
$1,\ \epsilon \to 1$
$0,\ 0 \to \epsilon$
$1,\ 1 \to \epsilon$
q_1
$\epsilon,\ \epsilon \to \$$
q_2
$\epsilon,\ \epsilon \to \epsilon$
q_3
$\epsilon,\ \$ \to \epsilon$
q_4

Equivalence of CFGs and PDAs

- To convert a CFG into a PDA
  - Basically use the stack to store partial derivations

Equivalence of CFGs and PDAs

- To convert a PDA into a CFG
  - See Theory of Computation book, section 2.2

Parsing context-free languages

- Bottom-up
  - CKY algorithm
- Top-down
  - Earley algorithm

The CKY algorithm

- S → NP VP
- S → VP
- NP → Det N
- NP → N
- VP → V NP
- VP → V
- V → book
- Det → that
- N → book
- N → flight

The CKY algorithm

- Base case
  - Fill in cells [i, i+1] with all possible nonterminals that can generate that word
- Recursive case
  - If A → BC and B is in cell [i, j] and C is in cell [j, k], fill in cell [i, k] with A

The Earley algorithm

S → NP VP

S → VP

NP → Det N

NP → N

VP → V NP

VP → V

V → book

Det → that

N → book

N → flight

The Earley algorithm

- Fill in a chart with dotted rules
  - Predictor
  - Scanner
  - Completer
- See SLP section 13.4.2 (2nd edition only)

The Earley algorithm

- Predictor
  - If the nonterminal after the dot is not a preterminal (POS tag), add a state for each possible expansion of the nonterminal

The Earley algorithm

- Scanner
  - If the nonterminal after the dot is a preterminal (POS tag), read the next word in the input and, if the word can be tagged as that tag, add a state to the next position in the chart

The Earley algorithm

- Completer
  - If the dot is at the end of the rule, add states corresponding to previous states that were “looking” for the rule to be completed, and advancing the dot

Natural languages are not context-free

- Why not?