CS 3313
Foundations of Computing: Equivalence of Pushdown Automata & Context Free Grammars

http://gw-cs3313-2021.github.io

Next: Equivalence of PDAs and CFLs

- Do PDAs accept Context free languages ?

- Prove: Given any CFG, there is a PDA that accepts the language generated by the grammar.
- Prove: Given any PDA, there is a CFG that generates the language accepted by the PDA.

Language of a PDA

- The common way to define the language of a PDA is by *final state*.
  - the set of all strings that cause the PDA to halt in a final state, after starting in q_0 with an empty stack.
  - The final contents of the stack are irrelevant
  - As was the case with nondeterministic automata, the string is accepted if any of the computations cause it to halt in a final state

- If M is a PDA, then L(M) is the set of strings w such that

$$(q_0, w, Z_0) \vdash^* (f, \lambda, \alpha) \text{ for final state } f \text{ and any } \alpha \in \Gamma^*$$

Language of a PDA – Alternate Definition

- Another way to define acceptance of a language by a PDA is by *empty stack*.
- If M is a PDA, then N(M) is the set of strings w such that

$$(q_0, w, Z_0) \vdash^* (q, \lambda, \lambda) \text{ for any state } q.$$

Equivalence of PDA Language Definitions

1. If L = L(P), then there is another PDA P' such that L = N(P').
2. If L = N(P), then there is another PDA P'' such that L = L(P'').

Either type of PDA acceptance works!

Automata and Grammars/Languages

- When we talked about closure properties of regular languages, it was useful to be able to jump between RE and DFA representations.
  - Similarly, CFG's and PDA's are both useful to deal with properties of the CFL's.
- Also, PDA's, being "algorithmic," are often easier to use when arguing that a language is a CFL and easier to use to design an algorithm to accept a language.
  - Example: It is easy to see how a PDA can recognize balanced parentheses; not so easy as a grammar.

Assumptions: Equivalence of PDAs and CFGs

- Every context free grammar has an equivalent grammar in Chomsky Normal Form
  - We provided the algorithm to convert a CFG to CNF Grammar
  - Using CNF, designed CYK (parsing) algorithm
- Every context free grammar has an equivalent grammar in Greibach Normal Form (GNF)
  - Similar algorithmic process exists for converting to GNF
  - GNF: every production is of the form $A \rightarrow a\alpha$ where $a \in T$, $\alpha \in V^*$
    - GNF derives strings using leftmost derivations
- Equivalence of PDAs and CFGs assumes (*without los of generality*) that the grammar is in GNF.

Derivations in GNF

- All productions of the form: $B \rightarrow a\alpha$ and $a \in T$ $\alpha \in V^*$
  - Rewrite as: $B \rightarrow aA_1\alpha_1$ whenever α is not empty.
- If we follow a leftmost derivation (production applied to leftmost variable in sentential form) then:

1. $S \rightarrow a_1 A_1\alpha_1$ where $a_1 \in T$, where $A_1 \in V$, $\alpha_1 \in V^*$
2. $S => a_1 A_1 \alpha_1 => a_1a_2 A_2 \alpha_2 \alpha_1$ where $A_1 \rightarrow a_2 \alpha_2$ and $\alpha_2 \in V^*$
3. $S => a_1 A_1 \alpha_1 => a_1a_2A_2 \alpha_2 \alpha_1 => a_1a_2 a_3A_3 \alpha_3 \alpha_2 \alpha_1$
   - where $A_2 \rightarrow a_3 A_3\alpha_3$ and $\alpha_3 \in V^*$
4. $\ldots S =>^* a_1a_2 a_3\ldots a_i A_i \alpha_i\ldots \alpha_2 \alpha_1$, where $a_i \in T$ and $\alpha_i \in V^*$
   - Leftmost derivation, at each step we generate terminal symbol a_i

Derivations in GNF and Moves in a PDA

- … $S \Rightarrow^* a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1$, where $a_i \in T$ and $\alpha_i \in V^*$
  - Leftmost derivation, at each step we generate terminal symbol a_i
- PDA reads input from left to right
  - It reads $a_1 a_2 a_3 \ldots a_i$ ….
- G derives: $S \Rightarrow^* a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1$
  - *Eventually* $a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1 \Rightarrow^* a_1 a_2 a_3 \ldots a_i x$
- *iff*
- PDA simulates $(q, a_1 a_2 a_3 \ldots a_i x, S) \vdash^* (q, x, A_i \alpha_i \ldots \alpha_2 \alpha_1)$

PDA for a Context Free Language

- Theorem: For every context free language L, there exists a PDA M such that L= L(M).
- Proof:
  - If L is a CFL then it is generated by some GNF grammar G=(V,T,P,S) with L(G)=L
  - Key idea: construct a PDA that simulates leftmost derivations in G
- PDA M= $(\{q_0,q_1,q2\}, T, V \cup \{z\}, \delta, q_0,\{q_2\})$
  - Stack alphabet = Set of Variables in G and the start stack symbol z
  - Alphabet = set of terminal symbols
  - $\delta(q_0, \lambda,z) = \{(q_1,Sz)\}$ /* push S to stack, goto q_1 and start simulation
  - $\delta(q_1, \lambda,z) = \{(q_2,z)\}$ /* if no input and 'empty stack' go to accept state
  - $\delta(q_1, a, A)$ contains (q_1, α) whenever $A \rightarrow a\,\alpha$ is a production in P
    - Simulate a derivation $A \Rightarrow a\,\alpha$

Proof – contd..

- Key idea: PDA reads a, pops A from stack, and pushes α to stack if $A \rightarrow a\,\alpha$ is a production in the grammar
- PDA simulates leftmost derivations in G
  - Input is processed left to right
- Prove: $S \Rightarrow^* x\,\alpha$ (using leftmost derivation) if and only if $(q_1, x, Sz) \vdash^* (q_1, \lambda, z)$
- Note: from definition of δ, $(q_0, x, z) \vdash (q_1, x, Sz)$
  - This starts PDA with S on TOS
- Proof by induction:
  1. If $(q_1, x, Sz) \vdash^* (q_1, \lambda, z)$ then $S \Rightarrow^* x\,\alpha$
  2. If $S \Rightarrow^* x\,\alpha$ then $(q_1, x, Sz) \vdash^* (q_1, \lambda, \alpha z)$

Proof: L(M) is in L(G)

If $(q_1, x, Sz) \vdash^* (q_1, \lambda, \alpha z)$ then $S \Rightarrow^* x\,\alpha$

- Induction on i the number of 'moves'/steps in PDA $\vdash^I$
- Basis: i=0..trivial
  - $(q_1, x, Sz) \vdash^0 (q_1, x, Sz)$ then $S \Rightarrow^0 S$
- Ind.: Assume IH holds for <i steps.
  - Now consider $(q_1, x, Sz) \vdash^i (q_1, \lambda, \alpha z)$, we can write $x=ya$ and
  - $(q_1, ya, Sz) \vdash^{i-1} (q_1, a, \beta z) \vdash (q_1, \lambda, \alpha z)$
  - From construction of PDA, $(q_1, a, \beta z) \vdash (q_1, \lambda, \alpha z)$ iff $\beta = A\gamma_1$ and $A \rightarrow a\gamma_2$ is a production in P (i.e., $\delta(q_1,a,A)$ contains (q_1, γ_2)).
  - From Ind.Hypo. $S \Rightarrow^* y\,\beta$, and from production $A \rightarrow a\gamma_2$ we have

    $$S \Rightarrow^* y\,\beta \Rightarrow ya\,\gamma_2\,\gamma_1 = x\,\alpha$$

Proof: L(G) is in L(M)

If $S =>^* x\,\alpha$ then $(q_1, x, Sz) \vdash^* (q_1, \lambda, \alpha z)$

- Induction on i the number of leftmost derivations in G
- Basis: i=0..trivial
  - then $S =>^0 S$ therefore $(q_1, x, Sz) \vdash^0 (q_1, x, Sz)$
  - Ind.: Assume IH holds for <i steps:
    - If $S =>^k x\,\alpha$ then $(q_1, x, Sz) \vdash^* (q_1, \lambda, \alpha z)$ for $k < i$
  - Now consider $S =>^{i-1} yA\,\gamma_1 => ya\,\gamma_2\,\gamma_1$
    - $A \to a\gamma_2$ is a production in P
  - From IH, we have: $(q_1, ya, Sz) \vdash^* (q_1, a, A\,\gamma_1\,z)$
  - From construction of PDA, $(q_1, a, A\,\gamma_1\,z) \vdash (q_1, \lambda, \gamma_2\,\gamma_1\,z)$ since $\delta(q_1,a,A)$ contains (q_1, γ_2).
  - Therefore $(q_1, ya, Sz) \vdash^* (q_1, \lambda, \gamma_2\,\gamma_1\,z)$....proved.
  - Finally note from construction of PDA: $(q_1, \lambda, z) \vdash (q_2, \lambda, z)$
  - Therefore $S =>^* w$ iff $(q_0, w, z) \vdash^* (q_2, \lambda, z)$
    - Substituting $\alpha = \lambda$

Example: PDA for Grammar

- $S \rightarrow aA$
- $A \rightarrow aABC \mid bB \mid a$
- $B \rightarrow b$
- $C \rightarrow c$

- $\delta(q_0, \lambda, z) = \{ (q_1, Sz)\}$ – startup the PDA, push S to stack
- $\delta(q_1, \lambda, z) = \{ (q_2, z)\}$ – accept if no input and empty stack
- $S \rightarrow aA$ therefore $\delta(q_1, a, S) = \{ (q_1, A)\}$
- $A \rightarrow aABC \mid bB \mid a$ therefore
  - $\delta(q_1, a, A) = \{ (q_1, ABC), (q_1, \lambda) \}$
  - $\delta(q_1, b, A) = \{ (q_1, B)\}$

Example:

- $S \rightarrow aA$ $A \rightarrow aABC \mid bB \mid a$ $B \rightarrow b$ $C \rightarrow c$
- Input $w = aabbc$ -- trace PDA on input w, and show leftmost derivation

PDA to CFG

- Theorem: If L =L(M) for a PDA M, then there is a context free grammar G such that L(G)=L(M)
- Proof: Read theorem 7.2 in textbook.
- Outline – given a PDA, we want to generate a grammar that simulates PDA via leftmost derivations
- The proof is rarely used to construct grammars – its purpose is to show the equivalence of the two formalisms CFG and PDA

CFG to PDA Conversion "Algorithm"

- The constructive proof can be implemented as an algorithm that takes a GNF Grammar G and generates a PDA
- We can then feed this PDA to a program that simulates/implements any PDA
    - We have an automated process for "writing" a parser!
- BUT…..the conversion/proof may lead to a non-deterministic PDA
    - We want our algorithms to be deterministic…i.e., parser should be deterministic
    - Question: Can we convert the grammar to a deterministic PDA ?

Deterministic Pushdown Automata

- A *deterministic pushdown automata (DPDA)*never has a choice in its move
- Restrictions on dpda transitions:
  - Any (state, symbol, stack top) configuration may have at most one (state, stack top) transition definition
  - If the DPDA defines a transition for a particular (state, λ, stack top) configuration, there can be no input-consuming transitions out of state s with a at the top of the stack
- Unlike the case for finite automata, a λ-transition does not necessarily mean the automaton is nondeterministic

Deterministic Context-Free Languages

- A context-free language L is *deterministic* (DCFL) if there is a dpda to accept L
- Sample deterministic context-free languages:

  $\{ a^n b^n : n \geq 0 \}$

  $\{ wcw^R : w \in \{a, b\}^* \}$
- Theorem: Deterministic and nondeterministic pushdown automata are not equivalent: there are some context-free languages for which no DPDA exists that accepts the language
  - *Syntax of most programming languages is deterministic context free*
- We will return to a discussion of DPDA and DCFLs after discussion of properties of CFLs

Next: Properties of Context Free Languages

- What are the properties of CFLs ?
- What types of languages are CFL ?
  - Can all properties/semantics of a programming language be captured by a CFL ?
  - Can natural languages be described by CFGs ?
    - Can we determine ambiguity and remove ambiguity ?
    - Can we parse natural languages using a CFG for the syntax ?
- If we combine CFLs using set operations, is the resulting language CFL ?
- How do we prove if a language is not context free ?
  - Pumping lemma for CFLs !!

Exercise: PDA for Grammar

- $S \to aSBBC \mid aBBC$
- $B \to b$
- $C \to cC \mid c$
- Input $w = abbcc$ -- trace PDA on input w, and show leftmost derivation