Previously:
Context-free grammars, for describing a more powerful class of languages

Today:
Pushdown automata for recognizing context-free languages.
Take-home Exam 1 due.

Then:
Assignment 4 out later today; due Tuesday after break.

Example of a CFG

- **A → aAb**
- **A → B**
- **B → ε**

- **Production rules**: substitutions
- **Non-terminals**: variable that can have a substitutions
- **Terminals**: symbols that are part of the alphabet, no substitutions
- **Start variable**: left side of top-most rule

Formal CFG Notation

- **Productions** = rules of the form \( \text{head} \rightarrow \text{body} \)
  - head is a variable
  - body is a string of zero or more variables and/or terminals
- **Start Symbol** = variable that represents “the language”
- **Notation**: \( G = (V, \Sigma, P, S) \)
  - \( V \) = variables
  - \( \Sigma \) = terminals
  - \( P \) = productions
  - \( S \) = start symbol
Pushdown Automata

- Add a stack to a FA
- Typically non-deterministic
- An automaton equivalent to CFGs

Schematic of a Finite Automaton

Schematic of a Pushdown Automaton

Notation

If at state $p$ with next input symbol $x$ and top of stack is $y$:
- go to state $q$ and replace $y$ by $z$ on stack

- $x = \epsilon$: ignore input, don’t read
- $y = \epsilon$: ignore top of stack and push $z$
- $z = \epsilon$: pop $y$
Example

Notation for transition diagrams:
\[ a, Z \rightarrow X_1X_2 \ldots X_k \]

- **Meaning**
  - On input \( a \), with \( Z \) on top of the stack
    - consume the \( a \)
    - make this state transition
    - replace the \( Z \) on top of the stack by \( X_1X_2 \ldots X_k \) (with \( X_1 \) at the top)

Formal PDA

\[ P = (Q, \Sigma, \Gamma, \delta, q_0, F) \]

- \( Q, \Sigma, q_0 \), and \( F \) have their meanings from FA
- \( \Gamma = \text{stack alphabet} \)
- \( \delta = \text{transition function} \)
  \[ \delta : Q \times \Sigma \times \Gamma \rightarrow P \times (Q \times \Gamma) \]
- Takes a state, input symbol (or \( \epsilon \)), and a stack symbol and gives you a finite number of choices of:
  1. A new state (possibly the same)
  2. A string of stack symbols (or \( \epsilon \)) to replace the top stack symbol

PDAs à la Sipser

- No intrinsic way to test for an empty stack
  - Get the same effect by initially putting a special symbol $ on the stack
    - Machine knows stack is empty if it sees $ during computation
- No intrinsic way to know when end of input string is reached
  - PDA achieves that effect because the accept state takes effect only when the machine is at the end of the input

N.B.: Different textbook authors use different conventions and notations

For \( ab^n \):

- \( q_1 \) = starting to see a group of as and bs
- \( q_2 \) = reading as and pushing as onto the stack
- \( q_3 \) = reading bs and popping as until the as are all popped
- \( q_4 \) = no more input and empty stack; accept

$ is a special "bottom of stack" symbol

$ is a special "bottom of stack" symbol
Instantaneous Descriptions (IDs)

- For a FA, the only thing of interest is its state
- For a PDA, we want to know its state and the entire contents of its stack
- Represented by an ID \((q, w, \alpha)\), where
  - \(q\) = state
  - \(w\) = waiting input
  - \(\alpha\) = stack [top on left; bottom on right]

Moves of the PDA

- If \(\delta(q, a, X)\) contains \((p, \alpha)\), then
  \((q, aw, X\beta) \vdash (p, w, \alpha\beta)\)
  - Extend to \(\vdash^*\) to represent 0 or more moves
  - Can subscript with the name of the PDA for clarity
  - Input string \(w\) is accepted if \((q_0, w, \varepsilon) \vdash^* (p, \varepsilon, \varepsilon)\) for any accepting state \(p\)
  - \(L(P) = \text{set of strings accepted by } P\)

Acceptance by Empty Stack

- Another one of those technical conveniences:
  - when we prove that PDAs and CFGs accept the same languages, it helps to assume that the stack is empty whenever acceptance occurs
  - \(N(P) = \text{set of strings } w\text{ such that}\)
    \((q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\) for some state \(q\)
  - Note \(q\) need not be in \(F\)
  - In fact, if we talk about \(N(P)\) only, then we need not even specify a set of accepting states

Example

\((q_1, \text{aabb}, \varepsilon) \vdash (q_2, \text{aabb}, \$)\)
\((q_2, \text{abb}, \text{a}$)\)
\((q_2, \text{ab}, \text{aa}$)\)
\((q_2, \varepsilon, \$)\)
\((q_2, \varepsilon, \varepsilon)\)
Example

• For our previous example, to accept by empty stack:

  1. **Add a new transition** \( \delta(p, \varepsilon, Z_0) = \{(p, \varepsilon)\} \)
     - That is, when starting to look for a new \(a-b\) block, the PDA has the option to pop the last symbol off the stack instead

  2. \(p\) is no longer an accepting state, in fact, there are **no accepting states**

Palindromes

**Input:**

\[ aaabcbaaa \]

Palindromes

**Input:**

\[ aaabcbaaa \]
Palindromes

Input: aaabcbaaa

Input: aaabcbaaa

Input: aaabcbaaa

Input: aaabcbaaa
Palindromes

Input: aaabcbaaa

Palindromes

Input: aaabcbaaa
**PDA Exercise**

The idea is to use the stack to keep count of the number of as and/or bs needed to get a valid string.

If we have a surplus of bs thus far, we should have corresponding number of as (two for every b) on the stack.

On the other hand, if we have a surplus of as, we cannot put bs on the stack since we can’t split symbols. So instead, put two “negative” a-symbols, where a negative a will be denoted by capital A.

Input:

`aaabcbaaa`

**ACCEPT!**

\[ L = \{ x \in \{a,b\}^* \mid n_a(x) = 2n_b(x) \} \]
Another PDA Exercise

Draw the PDA acceptor for

\[ L = \{ a^ib^jc^k \mid i = j + k \} \]

Equivalence of Acceptance by Final State and Empty Stack

- A language is \( L(P_1) \) for some PDA \( P_1 \) if and only if it is \( N(P_2) \) for some PDA \( P_2 \).
- Can show with constructive proofs
Final State ⇒ Empty Stack

Given \( P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) \), construct \( P_2 \):

1. Introduce new start state \( p_0 \) and new bottom-of-stack marker \( X_0 \).
2. First move of \( P_2 \): replace \( X_0 \) by \( Z_0X_0 \) and go to state \( q_0 \). The presence of \( X_0 \) prevents \( P_2 \) from “accidentally” emptying its stack and accepting when \( P_1 \) did not accept.
3. Then, \( P_2 \) simulates \( P_1 \), i.e., give \( P_2 \) all the transitions of \( P_1 \).
4. Introduce a new state \( r \) that keeps popping the stack of \( P_2 \) until it is empty.
5. If (the simulated) \( P_1 \) is in an accepting state, give \( P_2 \) the additional choice of going to state \( r \) on \( \varepsilon \) input, and thus emptying its stack without reading any more input.

Empty Stack ⇒ Final State

Given \( P_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) \), construct \( P_1 \):

1. Introduce new start state \( p_0 \) and new bottom-of-stack marker \( X_0 \).
2. First move of \( P_1 \): replace \( X_0 \) by \( Z_0X_0 \) and go to state \( q_0 \). Then, \( P_2 \) simulates \( P_1 \), i.e., give \( P_2 \) all the transitions of \( P_1 \).
3. Introduce a new state \( r \) for \( P_1 \), it is the only accepting state
4. \( P_1 \) simulates \( P_2 \)
5. If (the simulated) \( P_1 \) ever sees \( X_0 \) it knows \( P_2 \) accepts so \( P_1 \) goes to state \( r \) on \( \varepsilon \) input

Deterministic PDAs

- The PDAs we are dealing with are almost invariably non-deterministic
- A DPDA never has a choice of move
  - \( \delta(q, a, Z) \) has at most one member for any \( q, a, Z \) (including \( a = \varepsilon \)).
  - If \( \delta(q, \varepsilon, Z) \) is nonempty, then \( \delta(q, a, Z) \) must be empty for all input symbols \( a \)
- Why care?
  - Parsers are DPDAs
  - Thus, the question of what languages a DPDA can accept is really the question of what programming language syntax can be parsed conveniently

Non-deterministic PDAs

- A non-deterministic PDA allows non-deterministic transitions (e.g., defines multiple possible moves for a given configuration)
- Nondeterministic PDAs are strictly stronger than deterministic PDAs
- In this respect, the situation is not similar to the situation of DFAs and NFAs
- Non-deterministic PDAs are equivalent to CFLs
Real compilers

- However, unambiguous, deterministic CFGs are complicated and too restricted
- **Real parsers cheat by looking ahead one token**
  - This places certain restrictions on the grammar, but not as many
    - **STAY TUNED**: Learn about LL(1) and LR(1) grammars in CMPU 331
  - Some ambiguities (e.g. dangling else) are easily handled with one token lookahead

Equivalence of Parse Trees, Leftmost, and Rightmost Derivations

- The following about a grammar $G = (V, \Sigma, P, S)$ and a terminal string $w$ are all equivalent:
  1. $S \Rightarrow^* w$ (i.e., $w$ is in $L(G)$).
  2. $S \Rightarrow^* w$
  3. $S \Rightarrow_{lm}^* w$
  4. There is a parse tree for $G$ with root $S$ and yield (labels of leaves, from the left) $w$.
- Obviously (2) and (3) each imply (1).

Parse Tree Implies LM/RM Derivations

- Generalize all statements to talk about an arbitrary variable $A$ in place of $S$.
  - Except now (1) no longer means $w$ is in $L(G)$.
- Induction on the height of the parse tree.
- **Basis**: Height 1: Tree is root $A$ and leaves $w = a_1, a_2, \ldots, a_k$.
- $A \Rightarrow_{lm}^* w$ must be a production, so $A \Rightarrow_{lm}^* w$ and $A \Rightarrow w$. 
**Induction:** Height > 1: Tree is root A with children = \(X_1, X_2, \ldots, X_k\).

Those \(X_i\)'s that are variables are roots of shorter trees.

- Thus, the IH says that they have LM derivations of their yields.

- **Construct a LM derivation of** \(w\) **from** \(A\) **by starting** with \(\frac{A \rightarrow^*}{lm} X_1X_2 \ldots X_k\), then using LM derivations from each \(X_i\) that is a variable, in order from the left.

- **RM derivation analogous.**

**Example**

- Consider derivation \(S \Rightarrow AS \Rightarrow AAS \Rightarrow AA\)
  \(\Rightarrow A1A \Rightarrow A10A1 \Rightarrow 0110A1 \Rightarrow 0110011\)
- Sub-derivation from \(A\) is: \(A \Rightarrow A1 \Rightarrow 011\)
- Sub-derivation from \(S\) is: \(S \Rightarrow AS \Rightarrow A \Rightarrow 0A1 \Rightarrow 0011\)
- Each has a parse tree, put them together with new root \(S\).

**Derivations to Parse Trees**

- Induction on length of the derivation.
- **Basis:** One step. There is an obvious parse tree.
- **Induction:** More than one step.
  - Let the first step be \(A \Rightarrow X_1X_2 \ldots X_k\).
  - Subsequent changes can be reordered so that all changes to \(X_i\) and the sentential forms that replace it are done first, then those for \(X_j\) and so on (i.e., we can rewrite the derivation as a LM derivation).
  - The derivations from those \(X_i\)s that are variables are all shorter than the given derivation, so the IH applies.
  - By the IH, there are parse trees for each of these derivations.
  - Make the roots of these trees be children of a new root labeled \(A\).

**Only-If Proof (i.e., Grammar \(\Rightarrow\) PDA)**

- Prove by induction on the number of steps in the leftmost derivation \(S \Rightarrow^* \alpha\) that for any \(x, (q, wx, S) \vdash^* (q, x, \beta)\), where
  1. \(w\beta = \alpha\)
  2. \(\beta\) is the suffix of \(\alpha\) that begins at the leftmost variable (\(\beta = \varepsilon\) if there is no variable).
- Also prove the converse, that if \((q, wx, S) \vdash^* (q, x, \beta)\) then \(S \Rightarrow w\beta\).
- Inductive proofs in book.
- As a consequence, if \(y\) is a terminal string, then \(S \Rightarrow^* y\) iff \((q, y, S) \vdash^* (q, \varepsilon, \varepsilon)\), i.e., \(y\) is in \(L(G)\) iff \(y\) is in \(N(A)\).