CFG example

• Arithmetic expressions over {+,*,(,),a}
  - (a + a) * a
  - a * a + a + a + a + a

• A CFG generating this language:
  \( <\text{expr}> \rightarrow <\text{expr}> * <\text{expr}> \)
  \( <\text{expr}> \rightarrow <\text{expr}> + <\text{expr}> \)
  \( <\text{expr}> \rightarrow (<\text{expr}>) | a \)

Parse Trees

• Easier way to picture derivation: parse tree

CFGs and parse trees

• Is this a good grammar for arithmetic expressions?
  - can group wrong way (+ precedence over *)
  - different grammar for same language can force correct precedence/grouping

Some facts about CFLs

• CFLs are closed under
  - union (proof?)
  - concatenation (proof?)
  - star (proof?)

• Every regular language is a CFL
  - proof?
**Theorem:** A language $L$ is recognized by a NPDA if and only if $L$ is described by a CFG.

**Proof of ($\Rightarrow$):** $L$ is described by a CFG implies $L$ is recognized by a NPDA.

1. We'd like to non-deterministically guess the derivation, forming it on the stack.
2. Then scan the input, popping matching symbols 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?

**Proof of ($\Leftarrow$):** $L$ is recognized by a NPDA implies $L$ is described by a CFG.

- 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.

- 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

**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

**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 $A_{\text{start, accept}}$ generates all of the strings in the language recognized by the NPDA.

NPDA, CFG equivalence

- NPDA $P = (Q, \Sigma, \Gamma, \delta, \text{start}, \{\text{accept}\})$
- CFG $G$:
  - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
  - start variable $A_{\text{start, accept}}$
  - productions:
    - for every $p, r, q \in Q$, add the rule
      $$A_{p,r} \rightarrow aA_{r,s}b$$

from state $p$, read $a$, push $d$, move to state $r$
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 # of steps in $P$'s computation
   - base case: 0 steps. starts and ends at same state $p$. only has time to read empty string $\varepsilon$. $\rightarrow$ must be production of form $A_{p,p} \rightarrow \varepsilon$.

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: $A_{p,q} \xrightarrow{a} A_{r,s} \xrightarrow{b} ayb \rightarrow x$

Pumping Lemma for CFLs

**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 $i \geq 0$, $uv^i x y^i z \in L$, 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 $w = a^p b^p c^p$
  - With $|vy| > 0$ and $|vxy| \leq p$.

CFL Pumping Lemma Example

- Possibilities:
  - $w = \text{aaaa...aaabb...bbccc...c}$
  (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

**Theorem:** the following language is not context-free:

\[ L = \{ xx \mid x \in \{0,1\}^* \} \]

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

CFL Pumping Lemma

**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. \( |vxy| \leq p \).

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

Some non-terminal must repeat on long path.

CFL Pumping Lemma

- Schematic proof:
CFL Pumping Lemma

- How large should pumping length \( p \) be?
- Need to ensure other conditions:
  \[ |vy| > 0 \quad |vxy| \leq p \]

- \( b = \text{max \# symbols on rhs of any production} \) (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 \( > h \))

- Let \( m \) be the \# of nonterminals in the grammar
- To ensure path of length at least \( m+2 \), require
  \[ |w| \geq p = b^{m+2} \]
- 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 \( \geq m+1 \) non-terminals and 1 terminal.

- 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 \( i \geq 0, \text{uvxy}iz \in \mathcal{L} \)
- Is \( |vy| > 0 ? \)
  - Smallest parse tree \( T \) ensures
  - Is \( |vxy| \leq p ? \)
  - Red path has length \( \leq m+2 \), so \( \leq b^{m+2} = p \) leaves