Lecture 13

CSE 431
Intro to Theory of Computation

After Turing

Restricted computing models

McCulloch & Pitts (1943)
Neural Nets (neurons to deep nets)
as models of brain

Kleene (1951)
Neural Nets

complicated

Chomsky (1956)
Backus-Naur

Chomsky Context-Free languages

also are Context-Sensitive languages

Rabin & Scott (1959)
Nondeterministic Finite Automata (NFA)
introduced nondeterministic machines

You really simplified Kleene’s Thm

Following this claim:

CFGs \equiv \text{Pushdown Automata (PDA)}

Nondeterministic automata with a stack
Note: Normally we think of only being able to see top symbol on the stack, but we can convert such a machine into one with that restriction.

Suppose \( u = u_1 \ldots u_k \), \( v = v_1 \ldots v_l \).

Thus, for every \( L \), there is a CFG \( G \) s.t. \( L = L(G) \) \( \Rightarrow \) there is a PDA \( M \) s.t. \( L = L(M) \).

We will only prove one direction (the other direction is much less practically important, and is tricky. Sipser's text has the best exposition of it I have seen).  

Thus, for every CFG \( G \), there is a PDA \( M \) s.t. \( L(G) = L(M) \).
Proof: We give two proofs.

**Top-down parser**

Given CFG $G$

with rule $A \rightarrow \omega$

and start symbol $S$

\[ \rightarrow \text{apply rule} \]

$3, A \rightarrow \omega$

$\rightarrow \text{match input} \]

$3, a \rightarrow \varepsilon$

accept state

Idea: M applies rule of derivation to variable at top of the stack, matching up terminal at top of stack to the input.

eq. $S \rightarrow (S) | \varepsilon$

$S \rightarrow (S) \rightarrow (C)$

Stack:

\[
\begin{align*}
\text{Input: } & ( ) \quad \varepsilon \quad (\text{val: } ( ) \cdots ) \\
\text{Stack: } & S \quad S \quad \varepsilon \quad \varepsilon \\
\end{align*}
\]

corresponds to top-down DFS of

[Diagram of top-down DFS]
Note: This is a non-deterministic since a single A may have many rules.

Hard to know which rule to apply since the PDA hasn't even read any of the input symbols that the rule is supposed to generate.

**Bottom-up Parser.**

Idea: Push symbols onto the stack and try to invert the CFC derivation to produce S alone on the stack (other than $.)

- **Note:** If input has symbols $a_1, a_2, a_3$, we get $a_1 a_2 a_3$ and we start popping the first symbol one-by-one. We get $a_1 a_2 a_3$ on the stack (the last one is in top). This reverses the string.

**Bottom-Up Parser**

- Push input symbol
- Invert rule
In general, still nondeterministic and need look-ahead to know which rule to invoke.

Programming languages are designed so that one doesn't need to look ahead to know what rule to invoke.

**Chomsky Normal Form Conversion**

Rules of form:

- \( A \rightarrow BC \), \( B, C \in V, B, C \neq s \)
- \( A \rightarrow a \)
- \( S \rightarrow s \)

**Problems for general rules:**

1. \( A \rightarrow s \) for \( A \neq s \)
2. Right-hand side of length \( \geq 1 \) contains \( s \)
3. Right-hand sides of length \( \geq 2 \)
   - Rules of form \( A \rightarrow B \) (unit rules)
   - \( S \) on RHS of a rule

We get rid of these step by step:

\[
S \rightarrow (s) | s S 1 s e
\]
1. Add new $S_0$ and rule $S_0 \rightarrow S$
   New start symbol
   $S_0 \rightarrow S$, $S \rightarrow (S) | \epsilon | S_0$

2. Add new var $U$ for each $a \in E$
   Replace $a$ in any right hand side
   with $U$ and add rule
   $U \rightarrow a$
   $S_0 \rightarrow S$, $S \rightarrow USUSSS\epsilon$, $U \rightarrow (, T, )$

3. Rules of length $\geq 2$
   Add a chain of new intermediate
   variables to break up into size 2
   $S \rightarrow S$, $S \rightarrow UV|SS\epsilon$, $U \rightarrow (, T, )$

4. Compute $E$ the set of variables that
   can produce $\epsilon$:
   If $A \rightarrow \epsilon$ is a rule add $A \in E$
   Repeat: if $A \rightarrow BC$ is a rule with
   $B, C \in E$ add $A \in E$

$E = \{S_0, S, I\}$

Add $S \rightarrow S$

$V \rightarrow T$

(c) If $S_0 \in E$ add rule $S_0 \rightarrow \epsilon$
Any time such a rule is used it can be replaced by (a), (b), (c) rules above.

\[ S \rightarrow Sl, S \rightarrow UV|SS|l, \quad U \rightarrow (, T \rightarrow), V \rightarrow ST/T \]

Note: we added a number of unit rules that might not have been there before.

(3) Get rid of unit rules:

Create a directed graph on variables where there is an edge

\[ A \rightarrow B \iff A \rightarrow B \text{ unit rule} \]

Notes can do rules that walk around this graph doing replacements, but eventually need to do a non-unit rule at one of the unit reachable.

\[ S \rightarrow Sl, S \rightarrow UV|SS|l, \quad U \rightarrow (, T \rightarrow), V \rightarrow ST/T \]

non-unit rules marked.

- Add all non-unit rules to any var that can reach this.
- Remove unit rules.
Graph: $s_0 \rightarrow s \rightarrow t \rightarrow u$

Final Grammar in Chomsky Normal Form:

- $s_0 \rightarrow uu|ss|\epsilon$
- $s \rightarrow uu|ss$
- $u \rightarrow (\rightarrow)$
- $v \rightarrow st|\epsilon$