1 Chaper 6: Simplification of Context-Free Grammars and Normal Forms

1.1 Methods for Transforming Grammars

So, initially we would like to make claims about a general context-free language. So, to refresh, what even is a *context-free language*? Well, it is a language in which there is an initial state, and every string w in L can be formed by logical operations on the string, which is specified by the grammar for the language. Thus, a regular language (one able to be defined by an nfa) is a subset of all context-free languages.

However, there seems to be one problem with context-free languages in general: How do we deal with the empty-string? Usually, we can remove the empty string without too much of a loss in functionality.

We can define a language without the string, and if it is necessary, add an extra state dedicated to just inserting it in case we need it.

1.1.1 Theorem 6.1

Basically this theorem says that if a CFL contains a production of the form

A $\rightarrow x_1$ B x_2

and B$\rightarrow y_1 \mid y_2$...

Then, we can replace A $\rightarrow x_1$ Bx_2 with A $\rightarrow x_1\ y_1x_2|$...

Then, the language accepted by this new grammar is equivalent to the language accepted by the original one.

This is easy to accept as we are just replacing one implication with another, A and B (or whatever they're called).

useless variables: A variable is called useful if it takes part in any derivation. (The book used the example of a varable that can be a terminal). Then, a variable is called useless if there is no way of reaching it from the initial state or if there is no way it can lead to a terminal state.

1.1.2 Theorem 6.2

Let G=(V,T,S,P) be a context-free grammar. Then there is an equivalent language with no useless variables.

λ***-production***: a lambda production is one where a state can produce an empty string. This will then cause the language to lead to a terminal.

1.1.3 Theorem 6.3

Let G be a CFL without λ in its language. Then, there is an equivalent language with no λ-productions. To make an equivalent language without λ: We need to see the productions that lead to a lambda; remove the lambda; and then in every other variable that leads to that variable, we add the mullabe productions. (We create a set of nullable states).

1.2 6.2:Two Important Normal Forms

1.2.1 Chomsky Normal Form

When we want to study ways of writiing a grammar, one of the important forms to do this is the Chomsky Normal Form. The Chomsky Normal Form is when a grammar produces productions with at most two symbols.

For example
$A \rightarrow BC$
$A \rightarrow a$

1.2.2 Greibach Normal Form

This normal form does not restrict the amount of symbols that can go on the right side of a production, instead we deal with the positions of the terminals.

1.3 6.3: A Membership Algorithm for Context Free Grammars

Previously we had talked about there being some algorithms to determine membership in a particular language. Our brute force attempt was just to literally go through all the possible productions and compare them to the string we wanted to check. However, there is extists a parsing algorithm that is able to do this as a function of the input length in linear time.

The algorithm we use is called the CYK algorithm and only works with Chomsky Normal Form. We basically go through every element in the string and are able to determine how to produce it.

2 Pushdown Automata

A *pushdown automata* refers to a type of automata that can describe context-free languages. As we saw previously, if we are checking a string that can be of an arbitrarily long length, then we would need to have infinite memory. However, we can use something like a stack to store information. This would be useful when we have to do some sort of pattern matching.

2.1 Nondeterministic Pushdown Automata

To visualize a pushdown automata, we could imagine every symbol of the input file being read by a control unit. The control unit could posses the information regarding the states and possible transitions regarding the symbol. On the other side of the control unit, we could have the stack, into which we put (by usual stack operations) the symbol generated.

Definition 7.1

A non-deterministic pushdown accepter can be described by the tuple :

$$M=\{Q,\Sigma, \Gamma, \delta, q_0, z, F\}$$

1. **Q**: The finite set of internal states of the control unit.
2. Σ: the input alphabet
3. Γ: the finite set of symbols of the stack alphabet.
4. δ: The cross product of Qx($\Sigma \cup \{\lambda\}$)xΓ, the transition function.
5. q_0: the initial state of the control unit.
6. **x**: the stack start symbol.
7. **F**: teh set of final states.

The definition for the delta is kind of complicated. Essentially, it means that the transition requires an input state from the set of input states Q, the current state of the control unit, and the symbol that is placed on the stack. All transitions need the stack to have a nonzero number of elements, because the current state from the stack is required in our transition function.

Then,for exmaple, the transition

$\delta(q_1$, a,b

= (q_2, cd), (q_3, λ)