Specification of Languages by Rules

We have specified the set $S$ of balanced strings of parentheses via a definition by recursion. An alternative definition can be formulated via so-called rules, as follows:

- $B \rightarrow \lambda$
- $B \rightarrow (BB)$
- $B \rightarrow (B)$

Formally, a rule is a pair of strings over an alphabet $V \cup \Sigma$.

In this example, $V = \{B\}$ and $\Sigma = \{(,\}\}$.

Only elements of $V$ may occur on the left-hand side of a rule. They are also called nonterminals. One of the nonterminals, in the example $B$, is distinguished as the start symbol.

The rules may be used to generate strings over $\Sigma$ by beginning with the start symbol and then repeatedly replacing nonterminals by corresponding right-hand sides until a string in $\Sigma^*$ is obtained.

For example, from $B$ we may obtain $(BB)$, then $((B)B)$ and finally $((\lambda)B)$ and finally $((\lambda)\lambda)$, which is equal to $(())$.

Context-Free Grammars

Let $\Sigma$ be an alphabet.

A (context-free) grammar $G$ for $\Sigma$, with start symbol $S$, is a finite subset of $V \times (V \cup \Sigma)^*$, where $V$ is a set disjoint from $\Sigma$ and $S \in V$.

The elements of $G$ are called rules and are written as $A \rightarrow_G u$ or $A \rightarrow u$. Elements of $V$ are called nonterminals; elements of $\Sigma$, terminals.

The set of rules in the above example is a context-free grammar. Other examples of such grammars are $G_1$, consisting of rules

- $S \rightarrow \lambda$
- $S \rightarrow SS$
- $S \rightarrow (S)$

and $G_2$, consisting of two rules

- $S \rightarrow \lambda$
- $S \rightarrow (SS)$

In both cases, $S$ is the start symbol (and the only nonterminal).

Languages and Replacement

We say that a string $v$ can be obtained from $u$ by replacement with $G$, and write $u \Rightarrow_G v$ or $u \Rightarrow v$, if there exist strings $x$ and $y$ in $(V \cup \Sigma)^*$ and a rule $A \rightarrow w$ in $G$ such that $u = xAy$ and $v = xwy$.

For example, we have

- $((S)S) \Rightarrow_G ((\lambda)S)$

and also

- $(OS) \Rightarrow_G (\lambda \lambda)$.

A sequence of replacement steps

$$u_0 \Rightarrow_G u_1 \Rightarrow_G \cdots \Rightarrow_G u_n$$

is called a derivation in $G$ of $u_n$ from $u_0$.

For example,

- $S \Rightarrow SS \Rightarrow S((S)) \Rightarrow S(()) \Rightarrow (S)(()) \Rightarrow ()(())$

and

- $S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()(() \Rightarrow ()(()$

are both derivations in $G_1$.

We also write $u \Rightarrow^n_G v$ if $v$ can be derived from $u$ in this way (by zero or more replacement steps).

Thus,

- $S \Rightarrow^2_G (())$.

Grammars as Language Generators

The language $L(G)$ generated by $G$ is defined to be the set

$$\{w \in \Sigma^* : S \Rightarrow^*_G w\},$$

where $S$ is the start symbol of $G$.

In other words, the language generated by a grammar is the set of all strings of terminals that can be derived from the start symbol.

For instance, the language generated by $G_1$ is the set $S_1$ and the language generated by $G_2$ is the set $S_2$. Consequently, $L(G_1) = L(G_2)$.

We will sketch a proof that $L(G_2) = S_2$ below, but first give some examples of grammars for specific languages.
Examples

Let $\Sigma$ be the alphabet $\{a, b\}$. Give grammars for the following languages.

1. $L = \Sigma^*$
2. $L = \emptyset$
3. The set of all strings in $\Sigma^*$ of even length.
4. $L = \{a^n b^n : n \in \mathbb{N}\}$
5. The set of all palindromes in $\Sigma^*$.
6. The set of all strings in $\Sigma^*$ with an equal number of $a$'s and $b$'s.
7. The set of all decimal strings that represent numbers divisible by three. (In this case $\Sigma = \{0, 1, \ldots, 9\}$.)
8. The set of all strings in $\Sigma^*$ with an even number of $a$'s.

Example

Give a grammar for the set of all decimal strings that represent numbers divisible by three.

The grammar below is based on the following observation:

An integer is divisible by three if the sum of its digits is divisible by three.

Let $\Sigma$ be the set of digits $\{0, 1, \ldots, 9\}$ and $G$ be the grammar with start symbol $S_0$ and all rules,

$S_i \rightarrow d$

where $d \in \Sigma$, $i \in \{0, 1, 2\}$, and $d \mod 3 = i$, as well as all rules

$S_i \rightarrow dS_j$

where $d \in \Sigma$, $i \in \{0, 1\}$, $j \in \{0, 1, 2\}$, and $(d+j) \mod 3 = i$.

The language $L(G)$ represented by this grammar is the set of all strings in $\Sigma^*$ that represent integers divisible by three.

1. $L(G_2) \subseteq S_2$

We have to show that every string $w$ in $L(G_2)$ is an element of $S_2$. This can be proved by induction on the length of the derivation generating $w$.

Induction basis. If $w$ can be derived from the start symbol $S$ of $G_2$ in one step, $S \Rightarrow_G w$, then $w = \lambda$ and, hence, $w \in S_2$.

Next suppose $n > 1$. We assume, as induction hypothesis, that each string that can be derived from the start symbol $S$ of $G_2$ by fewer than $n$ replacement steps is an element of $S_2$. We need to show that each string that can be derived from $S$ in $G_2$ by $n$ steps is also an element of $S_2$.

Let $w$ be any arbitrary such string. Then there is a derivation

$S \Rightarrow_G \, w_1 \Rightarrow_G \cdots \Rightarrow_G \, w_n$

where $w = w_n$. Since $n > 1$ we must have $w_1 = (S) S$. In other words, the derivation is of the form

$S \Rightarrow_G (S) S \Rightarrow_G \cdots \Rightarrow_G (x) S \Rightarrow_G (x) y$,

where $x$ and $y$ are strings that can be derived from $S$ in fewer than $n$ steps. By the induction hypothesis, $x$ and $y$ are elements of $S_2$. By the definition of $S$, the string $w = (x) y$ is also an element of $S_2$.

2. $S_2 \subseteq L(G_2)$

We have to show that every string $w$ in $S_2$ can be derived from the start symbol $S$ of $G_2$. This assertion can be proved by induction on the number of applications of recursion needed to produce $w$ according to the definition of $S_2$.

Induction basis. The only string $w \in S_2$ that can be obtained without any application of the recursive rule is the empty string $\lambda$, which can be derived from $S$ in a single step, $S \Rightarrow_G \lambda$.

Next suppose $n > 0$. We assume, as induction hypothesis, that any string in $S_2$ that can be obtained by fewer than $n$ applications of the recursive rule can be derived from $S$ in $G_2$. We need to show that each string in $S_2$ that can be obtained by $n$ applications of the recursive rule can also be derived from $S$ in $G_2$. Let $w$ be any arbitrary such string.

Since $w$ requires at least one application of recursion, there exist strings $x$ and $y$ in $S_2$ such that $w = (x) y$ and $x$ and $y$ require fewer than $n$ applications of the recursive rule. By the induction hypothesis, there are derivations

$S \Rightarrow_G^* x$ and $S \Rightarrow_G^* y$.

But then there is also a derivation

$S \Rightarrow_G (S) S \Rightarrow_G (x) S \Rightarrow_G (x) y$,

which shows that $w \in L(G_2)$. 
Derivations

We have seen that the same string can possibly be generated in different ways, i.e., by different derivations, in a grammar.

For example, recall the grammar $G_1$ with rules

$$S \rightarrow \lambda$$
$$S \rightarrow SS$$
$$S \rightarrow (S)$$

and start symbol $S$.

The string $()()$ can be derived in five steps by

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()()$$

or

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ()()$$

in $G_1$.

The string $()()()$ can be derived by

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()SS \Rightarrow ()(S)S \Rightarrow ()()S$$

. $\Rightarrow ()()S \Rightarrow ()()()$ in $G_1$.

but also by

$$S \Rightarrow SS \Rightarrow SS \Rightarrow (S)S \Rightarrow SS \Rightarrow ()(S)S$$

. $\Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ in $G_1$.

Parse Trees

Derivations are sequences of replacement steps but can also be represented in a more abstract way by labelled trees.

A parse tree for a grammar $G$ is a labelled tree $T$, where (i) each leaf of $T$ is labelled by an element of $\Sigma$ or by the empty string $\lambda$, (ii) each interior (i.e., non-leaf) node of $T$ is labelled by an element of $V$, and (iii) for each node $i$ labelled by an element $A$ of $V$ there exists a rule $A \rightarrow x_1x_2\ldots x_n$, such that the children of $i$ are labelled by $x_1,x_2,\ldots,x_n$ (in this order).

For example,

$$\begin{array}{c}
S \\
| \\
S \\
| \\
S \\
| \\
e \\
\end{array}$$

is a parse tree for the grammar $G_1$ (where $e$ denotes $\lambda$).

The string one obtains by concatenating the labels of leaves from left to right is called the yield of the tree.

Parse Trees (cont.)

The yield of the parse tree in the above example is the string $()()$.

The tree can be constructed by starting with a single node labelled by $S$ and then expanding it in several steps by adding each time children to a leaf according to one of the grammar rules.

In this sense the tree represents the derivations of $()()$ we had shown earlier. The constructions results in the same tree for both derivations because they employ the same replacement steps, only in a different order.

One can argue that the derivations are therefore essentially the same and that the tree representation captures their essence in a more abstract way.

Ambiguous Grammars

If we construct parse trees from the two derivations of $()()()$ we obtain two different trees,

$$\begin{array}{c}
S \\
| \\
S \\
| \\
S \\
| \\
e \\
\end{array}$$

and

$$\begin{array}{c}
S \\
| \\
S \\
| \\
S \\
| \\
e \\
\end{array}$$
This indicates that the grammar $G_1$ is ambiguous, as the structure of the string $(())$ is not uniquely determined by the rules of $G_1$.

The grammar $G_2$, on the other hand, which defines the same language as $G_1$, is unambiguous and therefore preferable to $G_1$.

Example

Let $L$ be the set of all strings in $(a,b)^*$ with an even number of $a$'s.

Let $R_1$ be the set of rules

$$
R_1: \begin{align*}
S &\rightarrow \lambda \\
S &\rightarrow bS \\
S &\rightarrow aA \\
A &\rightarrow bA \\
A &\rightarrow aS
\end{align*}
$$

and $R_2$ the set of rules

$$
R_2: \begin{align*}
S &\rightarrow \lambda \\
S &\rightarrow bS \\
S &\rightarrow Sb \\
S &\rightarrow aSa
\end{align*}
$$

Let $S$ be the start symbol in each case.

Which of these grammars generate $L$?

Answer. All except the grammar based on $R_3$. For example, the string $abab$ can not be derived from $S$ by $R_3$.

We sketch next a proof that $L(R_1) = L$.

Lemma. For each string $w$ in $(V \cup \Sigma)^*$, if $S \Rightarrow^*_1 w$ then $w$ contains an even number of $a$'s, and if $A \Rightarrow^*_1 w$ then $w$ does not contain an even number of $a$'s.

Proof sketch. By induction on the length of the derivation of $w$.

We prove that for all $n > 0$, if $w$ is a string in $(V \cup \Sigma)^*$, then

(i) if $S \Rightarrow^*_1 w$ then $w$ contains an even number of $a$'s and

(ii) if $A \Rightarrow^*_1 w$ then $w$ does not contain an even number of $a$'s.

Let $n$ be an arbitrary, but fixed integer with $n > 0$.

Induction hypothesis. If $k < n$ and $w$ is a string in $(V \cup \Sigma)^*$ then $w$ contains an even number of $a$'s, provided

$S \Rightarrow^*_1 w$, and $w$ does not contain an even number of $a$'s, provided

$A \Rightarrow^*_1 w$.

Induction step. We prove the above assertion for all strings $w$ in $(V \cup \Sigma)^*$ for which $S \Rightarrow^*_1 w$ or $A \Rightarrow^*_1 w$, distinguishing several subcases depending on the first replacement step in the derivation.
**Example - Non-Context-Free Grammar**

Consider a grammar for a language over the alphabet $\Sigma = \{a\}$, with nonterminals $\{S, D, R, T, [,]\}$, start symbol $S$, and the following rules:

- $S \rightarrow [Da]$,
- $S \rightarrow a_0$,
- $Da \rightarrow aaD$,  
- $D[ \rightarrow R]$,
- $D[ \rightarrow T$,
- $aR \rightarrow Ra$,  
- $[R \rightarrow ]D$,
- $aT \rightarrow Ta$,  
- $[T \rightarrow e]$

This grammar, which is not context-free, generates the language $\{a^n : n \geq 0\}$.

The variable $D$ is used to double the length of a string of $a$'s:

$Dd^k \Rightarrow a^{2k}D$.

The symbols $[ $ and $] $ are used as left and right markers between which the generation of a string $a^n$ takes place. The variable $R$ initiates another application of doubling, whereas $T$ is used to terminate the process.