Languages

- A language is a set (usually infinite) of strings, also known as sentences.
- Each string consists of a sequence of symbols taken from some alphabet.
- An alphabet, $V$, is a finite set of symbols, e.g. $V = \{a, b, c, \ldots, z\}$.
- A string, $\omega$, in a language is a finite set of symbols from an alphabet.
  $$\omega = v_1 v_2 \ldots v_k \quad \forall \ v_k \in V$$
- The length of a string $\omega$, written $|\omega|$ is the number of symbols in it. The empty string is represented as $\epsilon$. $|\epsilon| = 0$.
- The set of all strings $\omega$ over an alphabet $V$ may be expressed as:
  $$\omega_k = \{ \omega \mid |\omega| = k \}$$
  $$\omega_0 = \{ \epsilon \}$$
  $$\omega = \bigcup \omega_k$$

Grammars

- If a language is an infinite set of strings, how can we define it?
  — we can’t just list all the strings
- One possibility is to have a finite set of rules for generating the strings
  — this is called a grammar
- Grammars have a number of syntactic categories:
  — Nonterminals – a set of symbols, $V_N$
  — Terminals – the “words” of the language (the alphabet) which cannot be broken down any further, $V_T$
  — Productions – the rules for the formation of sentences, $P$.
  — Start symbol – the starting point, $S$. $S \in V_N$
- Together these four things represent a grammar.
- A grammar, $G$, is a quadruple $(V_N, V_T, P, S)$ where $V_N \cap V_T = \emptyset$ i.e. non-terminals and terminals are disjoint.
  Note, the vocabulary $V_N \cup V_T = V$ (where $V_T$ is the alphabet).

An Example Grammar

- A simple example grammar is:
  $$G = ( \{ S, B \}, \{ a, b, c \}, \{ S \to aB, S \to bB, S \to cB, B \to a, B \to b, B \to c \}, S )$$
- In order to derive strings in $G$, we start with the goal symbol
  $$S \Rightarrow^* bc$$
  - $\Rightarrow$ is pronounced “directly derives” and represents one step in the derivation.
  - Sometimes it is useful to abbreviate the derivation by using the Kleene star:

$$L(G) = \{ \omega \mid \omega \in V_T^* \text{ and } S \Rightarrow^* \omega \}$$
**Derivations**

- Recall
  \[ \alpha_1 \Rightarrow_G^* \alpha_m \]
  if and only if
  \[ \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \ldots \Rightarrow_G \alpha_m \]
  and
  \[ L(G) = \{ \omega \mid \omega \in V_T^* \text{ and } S \Rightarrow_G^* \omega \} \]
- \( \omega \in V^* \) such that \( S \Rightarrow_G^* \omega \) is sometimes known as a sentential form or a sentence.
- For \( \omega \in L(G) \), the sequence of sentential forms from \( S \) to \( \omega \) is known as the derivation of \( \omega \).
- Note that we’ve seen this already – these derivations are paths in our Parse Tree or Abstract Syntax Tree.

**An Example**

- Grammar G2.1+:
  \[
  \begin{align*}
  &<E> ::= <T> | <E> + <T> \\
  &<T> ::= <F> | <T> * <F> \\
  &<F> ::= <X> | ( <E> ) \\
  &<X> ::= a | b | c
  \end{align*}
  \]
- Derive “a+b” from \( E \):
  \[
  \begin{align*}
  &E \Rightarrow E + T \\
  &\Rightarrow T + T \\
  &\Rightarrow F + T \\
  &\Rightarrow X + T \\
  &\Rightarrow a + T \\
  &\Rightarrow a + F \\
  &\Rightarrow a + X \\
  &\Rightarrow a + b
  \end{align*}
  \]
  leftmost derivation
  \[
  \begin{align*}
  &E \Rightarrow E + T \\
  &\Rightarrow E + F \\
  &\Rightarrow E + X \\
  &\Rightarrow E + b \\
  &\Rightarrow T + b \\
  &\Rightarrow F + b \\
  &\Rightarrow X + b \\
  &\Rightarrow a + b
  \end{align*}
  \]
  rightmost derivation

**Drawing Derivation Trees**

1. Each node in the tree is labelled with a symbol in \( V \).
2. The root node is always labelled with the start symbol of the language.
3. Leaf nodes have a label \( \in V_T \) if the derivation tree is complete.
4. If a node is not a leaf node, then its label \( \in V_N \)
5. If a node \( n \) labelled \( A \) has descendents:
   - \( n_1, n_2, \ldots, n_m \) labelled \( A_1, A_2, \ldots, A_m \)
   - then there must be a production:
     \[ A \rightarrow A_1 A_2 \ldots A_m \]
   - in the production rules.

**Example**

- Now \( G=(V_N, V_T, P, E) \)
  \[ \begin{align*}
  &V_N = \{ E, T, F, X \} \\
  &V_T = \{ a, b, c, +, \cdot, (,) \} \\
  &P = \ldots \text{there are 9 of them...}
  \end{align*} \]

- We can only draw derivation trees for type 2 and 3 Chomsky grammars. Type 0 and 1 required a directed graph to express them. See handout.
Derivations

- Derivations may be partitioned into **leftmost** derivations and **rightmost** derivations.

**Leftmost Derivation**

- With leftmost derivation we always expand the leftmost nonterminal in the sentential form. Leftmost derivation is often written as:
  \[ \alpha \Rightarrow \text{lm} \beta \]

**Rightmost Derivation**

- With rightmost derivation we always expand the rightmost nonterminal in the sentential form. Rightmost derivation is often written as:
  \[ \alpha \Rightarrow \text{rm} \beta \]

Ambiguity

- A context free (type 2) grammar is said to be ambiguous if there are one or more sentences \( \omega \in L(G) \), with two or more distinct leftmost (rightmost) derivations.
- An unambiguous grammar has only one parse tree for each sentence in the language.
- Ambiguity is bad – the Pascal dangling else problem:

  \[
  \begin{align*}
  S &\rightarrow \text{if } B \text{ then } S \\
  S &\rightarrow \text{if } B \text{ then } S \text{ else } S \\
  S &\rightarrow \text{statement} \\
  B &\rightarrow \text{condition}
  \end{align*}
  \]

  \[
  \begin{align*}
  S &\rightarrow \text{if } \text{condition} \text{ then } S \\
  S &\rightarrow \text{if } \text{condition} \text{ else } S \\
  S &\rightarrow \text{statement} \\
  B &\rightarrow \text{condition}
  \end{align*}
  \]

The Two Trees that result

- The Pascal dangling else problem was resolved by *defining* the *else* to bind to the most recent *if*.
- Note how the two different pretty-print formats *both* look plausible.
- Note how the alternatives *have different semantics* (assuming that semantics is based on syntax)
  — Just what is wanted when you write code to control your nuclear power plant!
- Ambiguous grammars can not be parsed using a recursive descent analyser
- We need to remove any ambiguity from the grammar before we start…
Different kinds of Grammars

- Grammars have a set of productions
- Most general form is $a \rightarrow b$
- Where $a$ is a string including a non-terminal and $b$ is any string (of terminals and non-terminals)

Different kinds of grammars constrain the productions:
- TYPE 1: $|a| \leq |b|$  
- TYPE 2: $a$ consists only of a non-terminal  
- TYPE 3: $a$ consists only of a non-terminal and $b$ consists only of a terminal or a terminal followed by a non-terminal

TYPE3 Grammars

- TYPE3 grammars have productions $a \rightarrow b$ where $a$ consists only of a non-terminal and $b$ consists of a terminal or a terminal followed by a non-terminal
- TYPE3 grammars are used to define terminal symbols of a programming language, e.g. numbers, identifiers
- TYPE3 grammars cannot define strings consisting of an arbitrary length matching strings, e.g. $a^n b^n$

$S \rightarrow aB_1$  
$S \rightarrow aA_1$  
$S \rightarrow aA_3$  
$B_1 \rightarrow b$  
$A_1 \rightarrow aB_2$  
$A_3 \rightarrow aA_2$  
$B_2 \rightarrow bB_1$  
$A_2 \rightarrow aB_3$  
$B_3 \rightarrow bB_2$

- Need approximately $2n$ non-terminals for strings $a^n b^n$

TYPE2 Grammars

- TYPE2 grammars have productions $a \rightarrow b$ where $a$ consists only of a non-terminal and $b$ consists of an arbitrary string (of terminals and non-terminals)
- TYPE2 grammars are used to define the syntax of a programming language, e.g. blocks, statements, expressions
- TYPE2 grammars can define strings consisting of an arbitrary length matching strings, e.g. $a^n b^n$

$S \rightarrow aBC$  
$S \rightarrow aXBC$  
$X \rightarrow aBC$  
$X \rightarrow aXBC$  
$CB \rightarrow BC$  -- reordering  
$aB \rightarrow ab$  
$bC \rightarrow bc$  -- context sensitive  
$bB \rightarrow bb$  
$cC \rightarrow cc$  -- context sensitive

- They cannot define strings $a^n b^n c^n$
- TYPE2 grammars cannot handle context constraints, such as definition and use of identifiers

TYPE1 Grammars

- TYPE1 grammars have productions $a \rightarrow b$ where $a$ includes a non-terminal and $|a| \leq |b|$  
- TYPE1 grammars can define strings consisting of an arbitrary length matching strings, e.g. $a^n b^n c^n$

$S \rightarrow aBC$  
$X \rightarrow aBC$  
$CB \rightarrow BC$  -- reordering  
$aB \rightarrow ab$  
$bC \rightarrow bc$  -- context sensitive  
$bB \rightarrow bb$  
$cC \rightarrow cc$  -- context sensitive

- TYPE1 grammars can handle context constraints, such as definition and use of identifiers
TYPE1 Grammars

- $S \rightarrow aBC$
- $X \rightarrow aBC$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$
- $S \Rightarrow aXBC$
- $X \rightarrow aXBC$
- $bC \rightarrow bc$
- $cC \rightarrow cc$
- $CB \rightarrow BC$
- $aB \rightarrow ab$
- $bB \rightarrow bb$
- $aB \rightarrow ab$
- $bC \rightarrow bc$
- $bB \rightarrow bb$
- $cC \rightarrow cc$

- reordering
- context sensitive

- Dead end
- no string generated

- A string of the language