UNRESTRICTED GRAMMARS

• Context-free grammar allows to substitute only variables with strings
• In an unrestricted grammar (or a rewriting system) one may substitute any non-empty string (containing variables and terminals) with another one (also with the empty string $\varepsilon$)

An unrestricted grammar is a 4-tuple $G = (V, \Sigma, R, S)$, where
• $V$ is the set of variables,
• $\Sigma$ is the set of terminals,
• $\Gamma = V \cup \Sigma$ is the alphabet of $G$,
• $R \subseteq \Gamma^* \times \Gamma^*$ is the set of rules, and
• $S \in V$ is the start variable

$(w, w') \in R$ is denoted as $w \rightarrow w'$

• Let
  • $G = (V, \Sigma, R, S)$,
  • strings $v, u, w, x \in \Gamma^*$ as well as
  • $v \rightarrow x$ a rule in $R$
  • $uvw$ yields string $uxw$ in grammar $G$,
    
    
    
    
    $uvw \Rightarrow_G uxw$
  • String $v$ derives string $w$ in grammar $G$,
    
    
    
    
    
    
    $v \Rightarrow_G W$,

    if there exists a sequence $v_1, v_2, ..., v_k \in \Gamma^*$ ($k \geq 0$) s.t.
    
    
    
    
    
    
    
    $v \Rightarrow_G v_1 \Rightarrow_G v_2 \Rightarrow_G ... \Rightarrow_G v_k \Rightarrow_G w$

• $k = 0$: $v \Rightarrow_G v$ for any $v \in \Gamma^*$
• $u \in \Gamma^*$ is a \textit{sentential form} of $G$ if $S \Rightarrow_G u$
• A sentential form consisting only of terminal symbols $w \in \Sigma^*$ is a \textit{sentence} of $G$
• The \textit{language of the grammar} $G$ consists of sentences
  \[ L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G w \} \]

The language \{ $a^k b^k c^k \mid k \geq 0$ \} is not a context-free one; it can be generated with an unrestricted grammar, which
1. Generates the variable sequence $L(ABC)^k$ (or $\varepsilon$)
2. Orders the variables lexicographically $\Rightarrow LA^k B^k C^k$
3. Replaces the variables with terminals

\[
S \rightarrow LT \mid \varepsilon \\
T \rightarrow ABCT \mid ABC \\
\quad \rightarrow aA \\
\quad \rightarrow ab \\
\quad \rightarrow BA \\
\quad \rightarrow AB \\
\quad \rightarrow CB \\
\quad \rightarrow BC \\
\quad \rightarrow CA \\
\quad \rightarrow AC
\]

$LA \rightarrow a$
$LA \rightarrow aa$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$
For example, we can generate the sentence $aabcc$ as follows:

$$S \Rightarrow LT \Rightarrow LABCT \Rightarrow LABCBC$$
$$\Rightarrow LABACBC \Rightarrow LAABCBC$$
$$\Rightarrow LAABBCC \Rightarrow aABBCC$$
$$\Rightarrow aaBBCC \Rightarrow aabBCC$$
$$\Rightarrow aabbCC \Rightarrow aabbC$$
$$\Rightarrow aabbcc$$

**Theorem.** A formal language $L$ generated by an unrestricted grammar can be recognized with a Turing machine.

**Proof.** Let $G = (V, \Sigma, R, S)$ be the unrestricted grammar generating language $L$. We devise a two-tape nondeterministic Turing machine $M_G$ for recognizing $L$.

$M_G$ maintains the input string on tape 1. On tape 2 there is some sentential form of $G$ which we try to rewrite as the input string.

At the beginning tape 2 contains the start variable $S$.

The computation of $M_G$ repeats the following stages.
1. The tape head of tape 2 is (non-deterministically) moved to some location on the tape;
2. we choose (non-deterministically) some rule of $G$ and try to apply it to the chosen location of the tape;
3. if the symbols on the tape match the symbols on the left-hand side of the rule, $M_G$ replaces them on tape 2 with the symbols on the right-hand side of the rule;
4. we compare the strings in tapes 1 and 2 with each other;
   a) if they are equal, the Turing machine enters the accepting final state and halts,
   b) otherwise, we go back to step 1.

---

**Theorem** If a formal language $L$ can be recognized with a Turing machine, then it can be generated with an unrestricted grammar.

**Proof.** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \text{accept}, \text{reject})$ be a standard Turing machine recognizing language $L$.

Let us compose an unrestricted grammar $G_M$ that generates $L$.

As variables of the grammar we take symbols representing all states $q \in Q$ of $M$. The configuration of the TM $uqa$ is represented as string $[uqa]$.

By the transition function of $M$ we give $G_M$ rules so that $[uqa] \Rightarrow_{G_M} [u'q'a'v'] \iff uqa \Rightarrow_M u'q'a'v'$.
Then

\[ x \in L(M) \iff [q_0, x] \Rightarrow G_M [u \text{accept} v], \ u, v \in \Sigma^* \]

There are three types of rules in \( G_M \):

1. Those that generate any string \( x(q_0, x), x \in \Sigma^* \) and \([, , q_0 \in V\) from the start variable:

\[
\begin{align*}
S &\rightarrow T[q_0] \\
T &\rightarrow \varepsilon \\
T &\rightarrow aTA_a \\
A_i\{q_0 \rightarrow \{q_{i-1}, A_n\} \\
A_i\{b \rightarrow bA_n\} \\
A_i\{ \rightarrow a\}
\end{align*}
\]

2. Those that simulate the transition function of the Turing machine:

\[
\begin{align*}
\delta(q, a) = (q', b, R) &\quad qa \rightarrow bq' \\
\delta(q, a) = (q', b, L) &\quad cqa \rightarrow q'cb \\
\delta(q, \square) = (q', b, R) &\quad q \rightarrow bq' \\
\delta(q, \square) = (q', b, L) &\quad cq \rightarrow q'cb \\
\delta(q, \square) = (q', \square, L) &\quad cq \rightarrow q'c
\end{align*}
\]
3. Those that replace a string of the form `[uacceptv]` to an empty string

\begin{align*}
\text{accept} & \rightarrow E_L E_R \\
aE_L & \rightarrow E_L \\
[E_L] & \rightarrow \varepsilon \\
E_R a & \rightarrow E_R \\
E_R] & \rightarrow \varepsilon
\end{align*}

Now a string $x$ in $L(M)$ can be generated as follows

$$S \Rightarrow_1 x[q_0x] \Rightarrow_2 x[uacceptv] \Rightarrow_3 x$$

### 3.3 The Definition of Algorithm

- The formulations of computation by Alonzo Church and Alan Turing were given in response to Hilbert’s tenth problem which he posed in 1900 in his list of 23 challenges for the new century.
- What Hilbert essentially asked for was an algorithm for determining whether a polynomial has an integral root.
- Today we know that this problem is algorithmically unsolvable.
- It is possible to give algorithms without them being exactly defined, but it is not possible to show that such cannot exist without a proper definition.
- In was not until 1970 that Matijasevič showed that no algorithm exists for testing whether a polynomial has integral roots.
• Expressed as a formal language Hilbert’s tenth problem is

\[ D = \{ p \mid p \text{ is a polynomial with an integral root} \} \]

• Concentrating on single variable polynomials we can see how the language \( D \) could be recognized
• In order to find the correct value of the only variable, we go through its possible integral values \( 0, 1, -1, 2, -2, 3, -3, \ldots \)
• If the polynomial attains value 0 with any examined value of the variable, then we accept the input
• A similar approach is possible when there are multiple variables in the polynomial

• For a single variable polynomial the roots must lie within

\[ \pm k \frac{c_{\text{max}}}{c_1}, \]

- where \( k \) is the number of terms in the polynomial,
- \( c_{\text{max}} \) is the coefficient with largest absolute value, and
- \( c_1 \) is the coefficient of the highest order term

• If a root is not found within these bounds, the machine rejects
• Matijasevič’s theorem shows that calculating such bounds for multivariable polynomials is impossible
• The language \( D \) can, thus, be recognized with a Turing machine, but cannot be decided with a Turing machine (may never halt)
Computability Theory

- We will examine the *algorithmic solvability* of problems
  - i.e. solvability using Turing machines

- We make a distinction between cases in which formal languages can be recognized with a Turing machine and those in which the Turing machine is required to halt with each input

- It turns out that there any many natural and interesting problems that cannot be solved using a Turing machine

- Hence, by Church-Turing thesis these problems are unsolvable by a computer!

**Definition 3.5** Call a language Turing-recognizable (or recursively enumerable, RE-language) if some Turing machine recognizes it.

**Definition 3.6** Call a language Turing-decidable (or decidable, or recursive) if some TM decides it (halts on every input, is total).

- The decision problem corresponding to language $A$ is *decidable* if $A$ is Turing-decidable.
- A problem that is not decidable is *undecidable*
- The decision problem corresponding to language $A$ is *semidecidable* if $A$ is Turing-recognizable
- Observe: an undecidable problem can be semidecidable.