Tik-79.148 **Spring 2001**
Introduction to Theoretical Computer Science
Tutorial 10
Solutions to demonstration exercises

4. A grammar (or unrestricted grammar) is a quadruple $G = (V, \Sigma, R, S)$, where V, Σ and S are defined in the same way as with context free grammars. The rules of a grammar are of the form

$$R \subseteq V^*(V - \Sigma)V^* \times V^*$$

There may be any string in the left side of a rule, as long as it contains at least one nonterminal symbol.

A solution grammar G for the exercise:

$$\begin{aligned}
\Sigma &= \{a\} \\
V &= \Sigma \cup \{S, [,], F, N, A\} \\
R &= \{S \to [N], N \to FNA, N \to e \\
&\quad FA \to AaF, Fa \to aF, F] \to], \\
&\quad [A \to [, [a] \to a[, [] \to e\}
\end{aligned}$$

The idea of the solution is to first derive n copies of F-nonterminals in the beginning of the word, and n copies of A-nonterminals in the end of the word. After that the F-symbols will be brought to the end of the word so that every time an A-symbol is passed, a terminal symbol a is added to the word. Since the word initially contains n copies of both the F- and A-symbols, eventually there will be $n \cdot n = n^2$ copies of a in the word. Finally, the symbol [will be moved to the end of the word and all the A-nonterminals are removed from the word at the same time.

For example, the word a^{3^2} can be derived as follows:

$$\begin{aligned}
S &\to [N] \to [FNA] \to [FFNAA] \to [FFFNAAA] \to [FFFAAA] \\
&\to [FFAaFAA] \to [FFAaAaFA] \to [FFAaAaAaF] \to [FFAaAaAa] \\
&\to [FAaFaAaAa] \to [FAaaFAaAa] \to [FAaaAaFaAa] \to [FAaaAaaFAa] \\
&\to [FAaaAaaAaFa] \to [FAaaAaaAaaF] \to [FAaaAaaAaa] \\
&\to [AaFaaAaaAaa] \to [AaaFaAaaAaa] \to [AaaaFAaaAaa] \\
&\to [AaaaAaFaaAaa] \to [AaaaAaaFaAaa] \to [AaaaAaaaFAaa] \\
&\to [AaaaAaaaAaFaa] \to [AaaaAaaaAaaFa] \to [AaaaAaaaAaaaF] \\
&\to [AaaaAaaaAaaa] \to [aaaAaaaAaaa] \to a[aaAaaaAaaa] \to aa[aAaaaAaaa] \\
&\to aaa[AaaaAaaa] \to aaa[aaaAaaa] \to aaaa[aaAaaa] \to aaaaa[aAaaa] \\
&\to aaaaaaa[Aaaa] \to aaaaaa[aaa] \to aaaaaaa[aa] \to aaaaaaaa[a] \\
&\to aaaaaaaaa[] \to aaaaaaaaa
\end{aligned}$$

5. If we want to prove that a problem is undecidable, we can either

   (i) suppose that the problem is decidable, and derive a contradiction with a chosen formalism, or

(ii) if we already know some other language is undecidable, we can try to reduce our own problem to the know one. Here we also have two options. First one is: we prove that if we could decide our own problem, we could use our solution to decide the undecidable problem. The other way is: we prove that in order to solve our own problem we should be able to solve the undecidable problem.

In this solution we use the latter way. The idea of the prove is to show: if we want to decide the problem "will an arbitrary Turing machine M halt on empty input", we first have to be able to decide the problem "will an arbitrary Turing machine M halt on input x". The latter is the general halting problem for Turing machines, which has been proved to be undecidable in the text book.

If the problem is decidable, there exists a Turing machine that decides the language

$$L = \{M \mid M \text{ halts on input} e\} \ .$$

In other words, there exists a Turing machine that can have encoding of any Turing machine as input, and decides whether the given machine would halt on empty input or not. To prove the original claim about the undecidability of the language it is enough to find one Turing machine, whose halting cannot be decided.

We construct a Turing machine M, which works in the following way: after having empty string as its input the machine first writes an arbitrary string x on the tape (the string can be chosen nondeterministically). After that the machine starts simulating an arbitrary Turing machine M', so that it does exactly the same transitions as the machine M' would do on the input x.

Now the machine M halts on the input e exactly when the machine M' halts on the input x. Because the latter problem is not decidable, there cannot exist any algorithm to solve whether M halts on input e. Hence, the problem is undecidable.

6. Let L_1 and L_2 be two given Turing acceptable languages. According to the definition, there exists two Turing machines M_1 and M_2 that accept (semidecide) these languages. (That is, $L(M_1) = L_1$ and $L(M_2) = L_2$). With the help of these machines, we construct 2-tape Turing machines $M_\cap$ and $M_\cup$ that accept the intersection and the union of the languages.

   (a) Intersection:
   In the beginning of the computation $M_\cap$ copies the input to the second tape. After that $M_\cap$ simulates the machine M_1 on input w using the first tape. If M_1 halts, we know that $w \in L_1$. Now we can simulate the machine M_2 on input w using the second tape. If also M_2 halts, the word belongs to both of the languages, and $M_\cap$ accepts the word. If w does not belong to either of the languages, the machine accepting that language will loop forever, and $M_\cap$ will not halt. Thus, $M_\cap$ accepts the language $L_1 \cap L_2$.

(b) Union:

The machine $M_\cup$ that accepts the union of the languages is constructed in the same way. However, here one must pay attention to the fact that the machines M_1 and M_2 cannot be simulated one after other. This is because if M_1 loops forever ($w \notin L_1$), the machine $M_\cup$ can never check whether $w \in L_2$.

The solution is to simulate the machines M_1 and M_2 parallel, one step at time. First the first step of computation of M_1 is performed, then the first step of M_2, then the second step of M_1, and so on. If the word belongs to either of the languages, either M_1 or M_2 will eventually halt and $M_\cup$ accepts the input.