Introduction to Computer Science

Géza Horváth

Tenth Lecture

Linear bounded automata – definition

Here we present a special, bounded version of the Turing machines, by which the class of context-sensitive languages can be characterized. This version of the Turing machine can work only on the part of the tape where the input is/was. These automata are called linear bounded automata (LBA).

Definition

Let $LBA = (Q, T, V, q_0, \#, \delta, F)$ *be a Turing machine, where*

$$\delta : Q \times (V \setminus \{\#\}) \to 2^{Q \times V \times \{Left, Right, Stay\}}$$

and

$$\delta : Q \times \{\#\} \to 2^{Q \times \{\#\} \times \{Left, Right, Stay\}}.$$

Then LBA is a (nondeterministic) linear bounded automaton.

Linear bounded automata – theorem

One can observe that # signs cannot be rewritten to any other symbol, therefore, the automaton can store some results of its subcomputations only in the space provided by the input, i.e., in fact, the length of the input can be used during the computation, only.

To establish a connection between the classes of context-sensitive languages and linear bounded automata we present the following theorem.

Theorem

The class of languages accepted by linear bounded automata and the class of context-sensitive languages coincide.

Deterministic linear bounded automata – open problem

Actually, there are other models of LBA, in which the workspace (the maximal tape-length during the computation) is limited by $c_1 \cdot |w| + c_0$, where w is the input and $c_0, c_1 \in \mathbb{R}$ constants. The accepting power of these models are the same as of those that have been presented.

However, the deterministic model is more interesting, since it is related to a long-standing and still open question.

It is known that every context-sensitive language can be accepted by deterministic Turing machines, using at most $c_2 \cdot |w|^2 + c_1 \cdot |w| + c_0$ space during the computations, where c_2, c_1, c_0 are constants. However, it is neither proven nor disproved that deterministic linear bounded automata (using $c_1 \cdot |w| + c_2$ space) can recognize every context-sensitive language. This is still a *famous* open problem.

Recursive and recursively enumerable languages

Now we introduce the recursive languages and the recursively enumerable languages. These two language classes are fundamental in computability theory. There are many equivalent definitions, however, we are going to use these two definitions now.

Definition

The language $L \in V^*$ is recursive, if there is an algorithm, which decides about any word $p \in V^*$, whether or not $p \in L$.

We can say a language L is recursive, if the word problem can be solved in L.

Recursive and recursively enumerable languages

We have defined the class of the recursively enumerable languages as languages which can be generated by unrestricted generative grammars. Now, we give another definition.

Definition

The language $L \in V^*$ is recursively enumerable, if there is a procedure, which specifies all the elements of L.

The two definitions of the recursively enumerable languages are equivalent. If there is a procedure, which specifies all elements of the language L, then there is a generative grammar which generates the language L, and if there is a generative grammar generating the language L, then this grammar itself is a procedure, which can be used to specify all elements of the language L.

Recursive and recursively enumerable languages

It is obvious that each recursive language is recursively enumerable, because we can list the words over an alphabet V, and select those words which are contained by the language L.

Theorem

The language L is recursive, if and only if both L and $\overline{L}$ are recursively enumerable.

Proof. First, we show that, if $L \in V^*$ and $\overline{L} = V^* \setminus L$ are recursively enumerables, then L – and also $\overline{L}$ – is recursive. The language L is recursively enumerable, so there is a procedure which lists the elements of L. Let us denote these words $p_1, p_2, \ldots$. However, $\overline{L}$ is recursively enumerable as well, so we have another procedure which lists the elements of $\overline{L}$. Let us denote these words $r_1, r_2, \ldots$. Now we can combine these two procedures, to use the first one, and then the second one, alternately. What we receive is the list of all words over the alphabet V: $p_1, r_1, p_2, r_2, \ldots$ and we know about each one if it belongs to the language L or not.

Recursive and recursively enumerable languages

Now, we show that if L is recursive, then L and $\overline{L}$ is recursively enumerable. We already mentioned that recursive languages are recursively enumerable, because we can list all the words over an alphabet V, and add the current word to the language if it is in L. The same algorithm can be used for the language $\overline{L}$, we can list the words again, and we add them to the language if they are not in the language L. QED.

Recursive and recursively enumerable languages

The following theorem shows the connection between the context-sensitive and recursive languages.

Theorem

Each context-sensitive language is recursive, but there are recursive languages which are not context-sensitive.

Proof. The first part of the theorem states that the word problem can be solved for each context-sensitive language.
For the second part, let us create the list of each possible context-sensitive generative grammar $G_i = (N_i, T_i, S_i, P_i)$, which generates numbers. (The set of the terminal letters of each grammar G_i are digits, so $T_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ for each G_i.)

Recursive and recursively enumerable languages

Now, we define language L, which contains the sequential numbers of grammars whose generated language does not contain its own sequential number (position in the list): $L = \{i \mid i \notin L(G_i)\}$.
We can create a list of all context-sensitive generative grammars which generates numbers, and we can decide whether or not a context-sensitive grammar generates its position in the list, so language L is recursive.
Now, suppose to the contrary that language L is context-sensitive. In this case, there is a context-sensitive grammar G_k, such that $L(G_k) = L$.
Then, by the definition of L, if $k \in L(G_k)$, then $k \notin L$ is a contradiction, and if $k \notin L(G_k)$, then $k \in L$ is also a contradiction. The only possible solution is that language L is not context-sensitive.
QED.

Recursive and recursively enumerable languages

The next theorem shows that there are languages which are not in the class of recursively enumerable languages, so the recursively enumerable language class does not contain all possible languages. The concept of the proof is similar to the previous proof.

Theorem

There exists a language L, which is not recursively enumerable.

Proof. Let us create the list of each generative grammar $G_i = (N_i, T_i, S_i, P_i)$ which generates numbers. (The set of the terminals of each grammar G_i are numbers: $T_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ for each grammar G_i). We have to note that it is easy to create an ordered list of all possible generative grammars which generates numbers.

Recursive and recursively enumerable languages

Now, we define language L, which contains the numbers of the grammars which does not generate the number of its position in the list:
$L = \{i \mid i \notin L(G_i)\}$.
Now, suppose to the contrary that the language L is recursively enumerable. In this case, there is a generative grammar G_k, such that $L(G_k) = L$. Then, by the definition of L, if $k \in L(G_k)$, then $k \notin L$ is a contradiction, and if $k \notin L(G_k)$, then $k \in L$ is also a contradiction. The only possible solution is that language L is not recursively enumerable. QED.

Recursive and recursively enumerable languages

We have already shown that the class of context-sensitive languages is a proper subset of the class of recursive languages. It has also been proven that the class of all languages is a proper superset of the class of recursively enumerable languages. Finally, we note that the complementer language of language L defined in the proof of the latest theorem is recursively enumerable, but not recursive. We can summarize our results in the following formula:

$$CS \subsetneq R \subsetneq RE \subsetneq AL$$

where CS stands for context-sensitive languages, R stands for recursive languages, RE stands for recursively enumerable languages, and AL stands for all languages.

Recursive and recursively enumerable languages

Theorem

The class of recursive languages is closed under complement operation.

Proof. A previous theorem states that language L is recursive, if and only if L and $\overline{L}$ are both recursively enumerable. However, we can apply the same theorem for the language $\overline{L}$, and what we receive as a result is that $\overline{L}$ is recursive, if $\overline{L}$ and L are both recursively enumerable, which is the same condition, so if L is recursive, then $\overline{L}$ is recursive, as well.
QED.

Recursive and recursively enumerable languages

Theorem

The class of recursively enumerable languages is not closed under complement operation.

Proof. The proof of this theorem is easy, we need only one example, where the language itself is recursively enumerable, and the complement language is not recursively enumerable. In the proof of a previous theorem we have defined a language L which is not recursively enumerable. The complement language $\overline{L}$ is recursively enumerable. Here we have a recursively enumerable language $\overline{L}$, whose complement is not recursively enumerable. QED.

Recursive and recursively enumerable languages

Theorem

The class of recursively enumerable languages is closed under regular operations.

Proof. We give a constructive proof here. Let the languages L_1 and L_2 be recursively enumerable. Let the grammar $G_1 = (N_1, T, S_1, P_1)$ such that $L(G_1) = L_1$, and let the grammar $G_2 = (N_2, T, S_2, P_2)$ such that $L(G_2) = L_2$. Without loss of generality we can suppose that $N_1 \cap N_2 = \emptyset$, and the terminal symbols appear only in rules having the form $A \to a$, where $A \in N,\ a \in T$. We define generative grammars G_{Un}, and G_{Co}, such that $L(G_{Un}) = L_1 \cup L_2$, and $L(G_{Co}) = L_1 \cdot L_2$.

Recursive and recursively enumerable languages

1. Union
   Let S be a new start symbol, such that $S \cap N_1 = S \cap N_2 = S \cap T = \emptyset$, and let

   $$G_{Un} = (N_1 \cup N_2 \cup \{S\}, T, S, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}).$$

2. Concatenation
   Let S be a new start symbol, such that $S \cap N_1 = S \cap N_2 = S \cap T = \emptyset$, and let

   $$G_{Co} = (N_1 \cup N_2 \cup \{S\}, T, S, P_1 \cup P_2 \cup \{S \to S_1 S_2\}).$$

Recursive and recursively enumerable languages

3. Kleene star
   In order to create a grammar generating the language $L(G_{Kl}) = L_1^*$ we use two grammars. Let the grammar $G_1 = (N_1, T, S_1, P_1)$, and let the grammar $G_2 = (N_2, T, S_2, P_2)$ such that $L(G_1) = L(G_2) = L_1 \setminus \{\lambda\}$, and $N_1 \cap N_2 = \emptyset$. Without loss of generality we can suppose that the terminal symbols appear only in rules having the form $A \to a$, where $A \in N,\ a \in T$. For the grammar G_{Kl} we again use a new start symbol S, where $S \cap N_1 = S \cap N_2 = S \cap T = \emptyset$. Then, let

   $$G_{Kl} = (N_1 \cup N_2 \cup \{S\}, T, S, P_1 \cup P_2 \cup \{S \to \lambda, S \to S_1, S \to S_1 S_2 S\}).$$

QED.

Recursive and recursively enumerable languages

Theorem

The class of recursively enumerable languages is closed under intersection.

Proof. By applying the definition of the recursively enumerable language, we can create a list of the elements of the recursively enumerable language L_1 without repetitions, and yet another list, which contains the elements from the recursively enumerable language L_2 without repetitions. Then, we can create a list, which contains one element from the list of the language L_1, and then one element of the list of the language L_2 alternately. We move an element from this combined list into the list of the $L_1 \cap L_2$, if the element appears twice.
QED.

Finally, we have to note that recursive languages are also closed under regular operations and intersection.

References

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.