Introduction to Computer Science

Géza Horváth

Sixth Lecture

Parsing

In formal language theory, parsing – or the so called syntactic analysis – is a process when the parser determines if a given string can be generated by a given grammar. This is very important for compilers and interpreters. For example, it is not too difficult to create a context-free grammar G_P generating all syntactically correct Pascal programs. Then, we can use a parser to decide – about a Pascal program written by a programmer – if the program is in the generated language $L(G_P)$. When the program is in the generated language, it is syntactically correct.

Earley algorithm – introduction

The Earley algorithm is designed to decide if a context-free grammar generates a terminal word. Sometimes it is not comfortable to create and use an equivalent Chomsky normal form grammar for a λ-free context-free grammar, because the Chomsky normal form grammar could have many more production rules than the original grammar. This is why the Earley algorithm is more widely used than the CYK algorithm for computerized lexical analysis. Although the Earley algorithm looks more complicated for humans, – and actually, it is more complicated compared to the the very simple CYK algorithm, – but after the implementation, there is no difference between the complexity of the two algorithms for computers.

Earley algorithm – description

Now we are going to show the steps of the λ-free version of the Earley algorithm. It can work with rules having form $A \to \lambda$ as well, with minor modification, but in practice we do not need the extended version.
Let $G = (N, T, S, P)$ be a λ-free, context-free grammar, and $p = a_1 a_2 \dots a_n \in T^+$, with integer $n > 0$. We are going to fill out the cells of an $(n+1) \times (n+1)$ triangular matrix M, except for the last cell $M(n, n)$. Over the cells of the first line of the matrix, we write the letters $a_1, a_2, \dots, a_n$, starting from the second cell and first letter, one after the other. The elements of the matrix are production rules from P, where the right hand side of each rule contains a dot character.

Earley algorithm – description

The triangular matrix M for the Earley algorithm:

	a_1	a_2	$\cdots$	a_{n-1}	a_n
(0,0)	(0,1)	(0,2)	$\cdots$	(0,n-1)	(0,n)
	(1,1)	(1,2)	$\cdots$	(1,n-1)	(1,n)
				$\vdots$	$\vdots$
				$\vdots$	$\vdots$
				(n-1,n-1)	(n-1,n)

Earley algorithm – description

The steps of the algorithm are the following:

1. Let $S \to .q \in M(0,0)$ if $S \to q \in P$, and let $j = 0$.
2. Let $A \to .q \in M(j,j)$ if $A \to q \in P$, and there exists an integer $k \leq j$ such that $B \to r.At \in M(k,j)$.
3. Let $j = j + 1$ and let $i = j - 1$.
4. Let $A \to ra_j.t \in M(i,j)$ if $A \to r.a_j t \in M(i,j-1)$.
5. Let $A \to rB.t \in M(i,j)$ if there exists an integer $i \leq k < j$ such that $A \to r.Bt \in M(i,k)$, and $B \to q. \in M(k,j)$.
6. If $i > 0$ then i=i-1 and goto 4.
   If $i = 0$ and $j < n$ then goto 2.
   If $i = 0$ and $j = n$ then finished.

Here $q \in (T \cup N)^+$, $A, B \in N$, $r, t \in (T \cup N)^*$, and of course i, j, k are integers.

Grammar G generates the word p ($p \in L(G)$), if and only if there is a production rule in M(0,n), whose left hand side is the start symbol S, and there is a dot at the end of the right hand side of the rule.

Earley algorithm – example

In this example, we have a λ-free context-free grammar G, and we have to decide if the word $a * a + a$ can be generated by this grammar.

$G = (\{S, A, B\}, \{a, +, *, (,)\}, S, P)$

$P = \{$
$S \to S + A$
$A \to A * B$
$B \to (S)$
$S \to A$
$A \to B$
$B \to a$
$\}$

	a	$*$	a	$+$	a
$S \to .S + A$ $S \to .A$ $A \to .A * B$ $A \to .B$ $B \to .(S)$ $B \to .a$	$B \to a.$ $A \to B.$ $S \to A.$ $A \to A. * B$ $S \to S. + A$	$A \to A * .B$	$A \to A * B.$ $S \to A.$ $A \to A. * B$ $S \to S. + A$	$S \to S + .A$	$S \to S + A.$ $S \to S. + A$
	-	-	-	-	-
		$B \to .(S)$ $B \to .a$	$B \to a.$	-	-
			-	-	-
				$A \to .A * B$ $A \to .B$ $B \to .(S)$ $B \to .a$	$B \to a.$ $A \to B.$ $A \to A. * B$

Bar-Hillel lemma – introduction

Although it is easy to find the exact position of a grammar in the Chomsky hierarchy, it is sometimes much more challenging to find the position of a language in the Chomsky hierarchy. The Bar-Hillel lemma is the first pumping – also called iteration – lemma, which gives properties shared by all context-free languages. Thus, if a language does not satisfy the conditions of the lemma, it is not context-free. This lemma – and its variations – can be used to show that a language is not context-free. On the other hand, languages satisfying the conditions may be not context-free.

Bar-Hillel lemma – theorem + proof

Theorem

(Bar-Hillel lemma) *For each context-free language L there exists an integer $n \geq 1$ such that each string $p \in L$, $|p| \geq n$ can be written in a form $p = uvwxy$, where $|vwx| \leq n$, $|vx| \geq 1$ and $uv^i wx^i y \in L$ holds for each integer $i \geq 0$.*

Proof. Let $L_1 = L \backslash \{\lambda\}$. It is ovious that if language L_1 satisfies the above conditions then language $L = L_1 \cup \{\lambda\}$ also holds the above conditions, so it is enough to prove the lemma for λ-free context-free languages.
We have already proved that each λ-free context-free language can be generated by a Chomsky normal form grammar. Further on let us assume that grammar G generating L_1 is in Chomsky normal form.

Bar-Hillel lemma – theorem + proof

Let us mark the number of nonterminals in grammar G by k, and let $n = 2^k + 1$. Let p be a word generated by grammar G, and let $|p| \geq n$. In this case, the height of the derivation tree of p is more than $k + 2$, where the last step is a nonterminal to terminal derivation. Let us investigate the last $k + 2$ height part of the longest path of the derivation tree. There must be a nonterminal A which appears twice, because the number of the nonterminals in G is less than the number of the nonterminals in this part. So there must be terminal words v, x such that $A \Rightarrow^* vAx$. Here $|vx| \geq 1$ because A has two different occurrences in the path, and the length of the generated word is increased by one in each derivation step, except for the derivation steps when we change a nonterminal to a terminal symbol.

Bar-Hillel lemma – theorem + proof

Also, there is a terminal word w, which can be derived from the last appearance of A on the path, so $A \Rightarrow^* w$ also holds.
Moreover, there are terminal words u, y such that $S \Rightarrow^* uAy$.
Based on these facts it is easy to show that
$S \Rightarrow^* uAy \Rightarrow^* uwy$
$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$
$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvvAxxy \Rightarrow^* uvvwxxy$
$\vdots$
This proves that $uv^i wx^i y \in L$ holds for each integer $i \geq 0$.
Finally, $|vwx| \leq n$, because the word vwx was derived with a derivation subtree of height at most $k + 2$, – where the last step was a nonterminal to terminal derivation, – so the length of the word vwx is maximum 2^k which is less than n.
QED.

Bar-Hillel lemma – example 1.

The following classical example shows an application of the Bar-Hillel lemma. We are going to prove that language $L = \{a^j b^j c^j \mid j \geq 0\}$ is not context-free.

In order to do this suppose to the contrary that language L is context-free. Let $j \geq (n/3)$, then, by the Bar-Hillel lemma, $a^j b^j c^j$ can be written in a form $uv^i wx^i y$ such that $|vx| \geq 1$ and $uv^i wx^i y \in L$ holds for each integer $i \geq 0$. First, neither of v nor x should contain two or more different letters, because repeating them would change the form of the words in L. So v is a unary word (some power of a letter, e.g. $aa...a$) and x is a unary word as well. In this case, when we increase the integer i, we change the number of one or two different letters, but we cannot change the number of each letter, which is a contradiction.

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.