The Circularity Problem for Attribute Grammars

Charles Gordon
CSE 531 – Automata, Fall 2000

1 An introduction to attribute grammars

Context-free grammars are used to specify the syntax of a language, but not its semantics. Knuth [3] devised attribute grammars as a mechanism for including the semantic rules of a language with its syntax. In an attribute grammar, each nonterminal symbol in the context-free grammar is augmented with a set of attributes. These attributes can represent just about any semantically meaningful concept, such as scalar value, length, memory location, polarity, etc. In addition to the attributes, each production rule is augmented with a set of semantic rules, which are evaluated to determine the semantic value of a particular sentence in the grammar. There are two types of attributes, synthesized and inherited. Synthesized attributes are determined by following the parse tree from the bottom to the top (i.e. children nodes determine the value of synthesized attributes in their parents). Inherited attributes are determined by following the tree from top to bottom. In the example that follows, synthesized attributes are notated with an up arrow (↑) and inherited attributes are notated with a down arrow (↓). The example in figure 1 is an adaptation from Knuth [3]. The subscript numbers are used only to denote different instances of the same nonterminal for explanatory purposes; they are not actually different nonterminals.

0: Number → Sign List	(i) List↓Scale := 0
	(ii) Number↑Value := IF Sign↑Neg THEN - List↑Value ELSE List↑Value
1: Sign → +	(i) Sign↑Neg := False
2: Sign → -	(i) Sign↑Neg := True
3: List → BinaryDigit	(i) BinaryDigit↓Scale := List↓Scale
	(ii) List↑Value := BinaryDigit↑Value
4: $List_0$ → $List_1$ BinaryDigit	(i) $List_1$↓Scale := $List_0$↓Scale + 1
	(ii) BinaryDigit↓Scale := $List_0$↓Scale
	(iii) $List_0$↑Value := $List_1$↑Value + BinaryDigit↑Value
5: BinaryDigit → 0	(i) BinaryDigit↑Value := 0
6: BinaryDigit → 1	(i) BinaryDigit↑Value := $2^{\text{BinaryDigit↓Scale}}$

Figure 1: A simple attribute grammar for signed binary numbers

Disregarding the semantic rules on the right, production rules zero to six define a grammar that accepts strings of binary digits (such as +01010101, or –100001). The added semantic rules actually compute the decimal value of the binary number. Notice that just about any well-defined function can be used in the semantic rules. In this case

we are using a conditional operator (rule 0i), addition (rule 4i and 4iii) and exponentiation (rule 6i). The semantic rules define the value of the synthesized attributes for the left-hand side nonterminal and the inherited values for the right-hand side nonterminals. For example, in rule zero the value attribute of number is set to the value of the binary digit list, modified by the sign value. The easiest way to understand what is going on in the parsing of a language using an attribute grammar is to look at the parse tree decorated with the attribute values. Figure 2 shows a parse tree for the string –101 using the attribute grammar in figure 1.

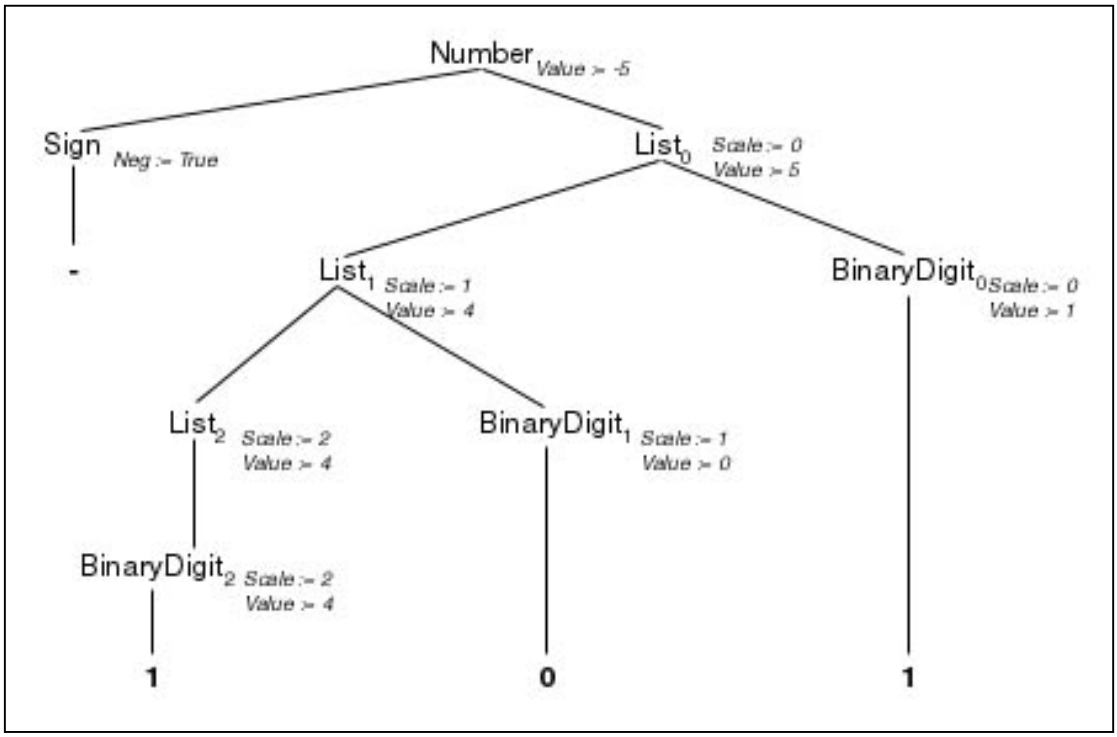

Figure 2: Parse tree for –101 using attribute grammar from figure 1.

Ignore the filled in values of the attributes for a moment and consider the way one would go about calculating them. Starting from the top, one would fill in the inherited values down to the leaves and then start back up the tree filling in synthesized values. In this case the first inherited attribute we run into is $List_0\downarrow Scale$, which is set to zero (from rule 0i). The next two are $List_1\downarrow Scale$, which is set to one (from rule 4i), and $BinaryDigit_0\downarrow Scale$, which is set to zero (from rule4ii). After that, $List_2\downarrow Scale$ is set to two (from rule 4i) and $BinaryDigit_1\downarrow Scale$ is set to one (from rule 4ii). Finally, $BinaryDigit_2\downarrow Scale$ is set to two (from rule 4ii). At this point we know the scales of all the binary digits, which allows us to calculate their actual values. Going back up the tree, $BinaryDigit_0$ is set to 2^0 (from rule 6i), $BinaryDigit_1$ is set to zero (from rule 5i) and $BinaryDigit_2$ is set to 2^2 (from rule 6i). The value attributes then propagate through the list nonterminals and up to number, where it is combined with the negative sign to give a final value of –5.

2 The circularity problem for attribute grammars

In order for an attribute grammar to be well formed the attributes associated with nonterminals at any node in the parse tree must be possible to evaluate using the semantic rules for the grammar. Non-trivial context-free grammars can produce an infinite number of trees, so the problem of determining their well-formedness is non-trivial. The principal problem is circularities in the dependency graph between attributes. For example, if an attribute a_1 depends on a_2, which in turn depends on a_1, there is no way to evaluate either a_1 or a_2. The best way to visualize this is to actually draw the dependency graph for a grammar. Figure 3 shows such a graph for the attribute grammar in figure 1.

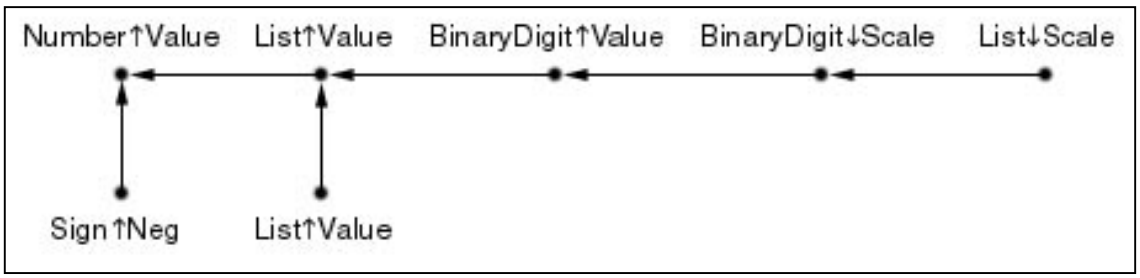

Figure 3: Attribute dependency graph for the grammar in figure 1.

The dependency graph is a combination of the individual dependencies from each rule in the grammar (which is why there are multiple List↑Value attributes). This particular graph has no circularities, which means that it is well formed (i.e. all possible parse trees generated by the grammar can be interpreted by the semantic rules). A circular dependency could be formed by reversing the arrow from Sign↑Neg to Number↑Value and adding an arrow from Sign↑Neg to the lower List↑Value. At that point any of the four attributes that made up the cycle would be impossible to evaluate. The circularity problem for attribute grammars is the detection of such cycles in the dependency graph of attributes.

3 Complexity of the circularity problem

Knuth [3] knew of the circularity problem and devised an algorithm that decided it in $O(2^{cn^2})$ steps. Jazayeri [1,2] was the first to prove that the problem was intractable however. He came up with the following theorem.

THEOREM. *The lower bound on the complexity of the circularity problem for attribute grammars is $2^{cn/\log n}$, where n is the size of the grammar description. That is, there is a constant $c > 0$ such that any correct algorithm must run for $2^{cn/\log n}$ steps for infinitely many n's.*

The original proof of this theorem in [1] used a reduction from the recognition problem for writing pushdown acceptors. The manipulation of the stack caused this particular proof to be very complicated. A second paper by Jazayeri [2] gave a much simpler, more elegant version of the proof using alternating Turing machines. More

specifically, the acceptance problem for alternating Turing machines was reduced to the circularity problem for attribute grammars. The acceptance problem is known to take exponential time in terms of deterministic Turing machines. Jazayeri gave a reduction that takes a machine M with input w and produces a grammar G(M,w) such that M accepts w iff G(M,w) is circular.

The key insight used by Jazayeri in his proof was that alternating Turing machines have a very natural correspondence with grammars. In particular, existential states can be used to select between different alternatives for a nonterminal expansion, and universal states can be used to evaluate a single right-hand side for a nonterminal. The reduction creates a nonterminal in the grammar for each state in M. The set of attributes associated with each nonterminal in the parse tree corresponds to the configuration of M at that step in the computation. In this way the grammar is used to simulate the action of the alternating Turing machine. The whole trick of the proof was designing the attributes so that reaching an accepting state corresponded to creating a dependency in the attribute graph. Jazayeri accomplished this using a very simple language ($\Sigma = \{a,b\}$) and carefully constraining the number of attributes available to each nonterminal so at least one had to be repeated. The reduction shows that the circularity problem is at least as complex as the acceptance problem, which is exponential in the size of the input.

4 References

1. Jazayeri, M., Ogden, W.F., and Rounds, W.C. The intrinsically exponential complexity of the circularity problem for attribute grammars. *Commun. ACM* 18(12): 697-706, Dec. 1975.

2. Jazayeri, M. A simpler construction for showing the intrinsically exponential complexity of the circularity problem for attribute grammars. *Commun. ACM* 28(4): 715-720, Oct. 1981.

3. Knuth, D.E. Semantics of context-free languages. *Math. Systems Theory*, 2:127-145, 1968.

4. Lemone, K. Course material on attribute grammars on the web. *http://penguin.wpi.edu:4546/course/cs4533/PLT6print.html.*