CS 131 Compilers: Discussion 7: Syntax Directed Translation & Mid Term Preparation1

杨易为 季杨彪 尤存翰
{yangyw,jiyb,youch}@shanghaitech.edu.cn

2021 年 4 月 12 日

1 Syntax Directed Translation

Syntax-directed translation refers to a method of compiler implementation where the source language translation is completely driven by the parser. In other words, the parsing process and parse trees are used to direct semantic analysis and the translation of the source program. This can be a separate phase of a compiler or we can augment our conventional grammar with information to control the semantic analysis and translation. Such grammars are called attribute grammars.

We augment a grammar by associating attributes with each grammar symbol that describes its properties. An attribute has a name and an associated value: a string, a number, a type, a memory location, an assigned register—whatever information we need. For example, variables may have an attribute "type" (which records the declared type of a variable, useful later in type-checking) or an integer constant may have an attribute "value" (which we will later need to generate code).

With each production in a grammar, we give semantic rules or actions, which describe how to compute the attribute values associated with each grammar symbol in a production. The attribute value for a parse node may depend on information from its children nodes below or its siblings and parent node above.

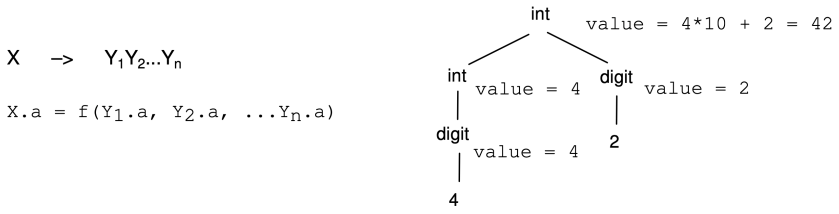
Consider this production, augmented with a set of actions that use the "value" attribute for a digit node to store the appropriate numeric value. Below, we use the syntax X.a to refer to the attribute a associated with symbol X.

```
digit    -> 0  {digit.value = 0}
         |  1  {digit.value = 1}
         |  2  {digit.value = 2}
         ...
         |  9  {digit.value = 9}
```

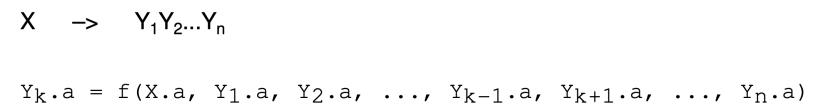
Attributes may be passed up a parse tree to be used by other productions:

```
int1     -> digit        {int1.value = digit.value}
         |  int2 digit   {int1.value = int2.value*10 + digit.value}
```

There are two types of attributes we might encounter: synthesized or inherited. Synthesized attributes are those attributes that are passed up a parse tree, i.e., the left- side attribute is computed from the right-side attributes. The lexical analyzer usually supplies the attributes of terminals and the synthesized ones are built up for the nonterminals and passed up the tree.



Inherited attributes are those that are passed down a parse tree, i.e., the right-side attributes are derived from the left-side attributes (or other right-side attributes). These attributes are used for passing information about the context to nodes further down the tree.



Bison is a little bit different, which is bigger than LALR(k).

2 Mid Term preparation

2.1 Regular Expressions

1. Write a regex that matches binary strings divisible by 8.
2. Provide aregulargrammar for the regex from part (a).

2.2 Finite State Automata

1. Write the corresponding NFA for the regular expression (0|1)?(10|01)+;
2. Convert the NFA from part (a) into a DFA.

2.3 Grammar Rewriting and LL(k) Parsing

$$
\begin{aligned}
S &: E \dashv \\
E &: E + E \\
&\mid E * E \\
&\mid \text{ ID}
\end{aligned}
$$

1. Show that the grammar is ambiguous with two different leftmost derivations of the string $a + b * c$;
2. Rewrite this grammar so that it preserves the standard order of operations, is LL(1),and is unambiguous. Draw the resulting tree for the string $a + b * c$;
3. Write down the equivalent unambiguous grammar that enforces both left associativity and correct precedence. Why can't this be achieved with an LL(1) grammar?

2.4 Earley's Algorithm

Consider the following grammar:

$$
\begin{array}{lcl}
P & : & E \dashv \\
E & : & ID \\
 & | & \lambda ID \,.\, E \\
 & | & E\ E
\end{array}
$$

1. Use it to parse the following expression with Earley's algorithm: λID . ID ID ⊣

	λ	ID	.	ID	ID	
	0	1	2	3	4	5
a						
b						
c						
d						
e						
f						
g						
h						
i						
j						

2. Is the expression in the language? Is the grammar ambiguous?