COMPILER CONSTRUCTION (Bottom-up & Shift-reduce parsing) Fall 2019

Lecture 18: Aug. 26,28 2019

Instructor: K.R. Chowdhary *: Professor of CS*

18.1 Bottom-up Parsing

A bottom-up parse corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top). For bottom up parsing we will use the grammar given below:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T \times F \mid F \\ F &\rightarrow (E) \mid num \mid id \end{aligned} \tag{18.1}$$

The sequence of snapshots $(a) - (d)$ in Fig. 18.1 are the steps for parsing the expression $id + id$.

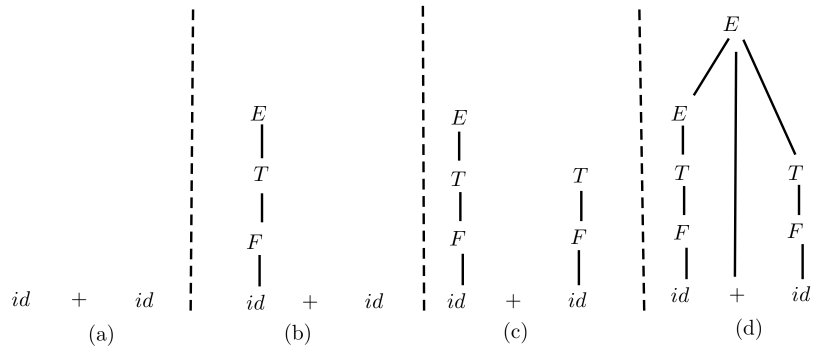

Figure 18.1: Steps for bottom-up parse for $id + id$

In the following we are going to discuss general type of bottom-up parsing, called as the shift-reduce parsing. Later we will discuss the LR grammar, which are the largest class of grammars, used for building the shift reduce parsers. The tools, called *automatic parser generators* are used for building the parsers. In the immediate next, we will discuss the basic principles of writing suitable grammars, that can be used with the parser generators.

18.1.1 Reductions

We can think of bottom-up parsing as the process of reducing the given string w (the sentence) to the start symbol of the grammar. This process is in reverse order of what is used in top-down parsing. At each reduction step, a substring that matches the *body* of a production (right hand side of a production rule) is reduced to the *head* of the production rule. The key decisions in the bottom-up parsing are:

1. when to reduce the body of a head by corresponding production, and
2. which among the matching rules to apply out of the given productions,

as the parsing proceeds. For example, in the bottom-up parsing shown in Fig. 18.1, we have following order of reductions:

$$id + id, \quad F + id, \quad T + id, \quad E + id, \quad E + F, \quad E + T, \quad E.$$

Note that, the application of production rules for carrying out the reduction from $id + id$, to $F + id$ is $F \to id$. Accordingly, the order of rules applied in reducing are: $F \to id$, $T \to F$, $E \to T$, $F \to id$, $T \to F$, $E \to E + T$.

18.1.2 Handle Pruning

The bottom-up parsing uses a left-to-right scan of the input, and constructs a *right most derivation* in reverse. Note that, when we parsed top-down, the right most part of the sentence used to be generated at end. In bottom-up, it is reverse, i.e., now the last generated will be reduced first. The substring that matches the body of a production is called "handle", and its reduction represents one step along the reverse of a right-most derivation. For example, adding subscripts to the tokens id for clarity, the handles during the parse of $id_1 + id_2$, according to the expression grammar (18.1), are shown in Table 18.1.

Table 18.1: Handles during the parse of $id_1 + id_2$

Right sentential form	Handle	Reducing production
$id_1 + id_2$	id_1	$F \to id$
$F + id_2$	F	$T \to F$
$T + id_2$	T	$E \to T$
$E + id_2$	id_2	$F \to id$
$E + F$	F	$T \to F$
$E + T$	$E + T$	$E \to E + T$

Finally, if $S \Rightarrow^*_{rm} \alpha A w \Rightarrow_{rm} \alpha\beta w$, as shown in Fig. 18.2, then the production $A \to \beta$ in the position following α is a *handle* of sentential $\alpha\beta w$. In other words, a handle of a right sentential form γ is a production $A \to \beta$ such that a position of γ where the string β may be found, such that replacing β at that position by A produces the previous right-sentential form in a right-most derivation of γ.

Notice that the string w to the right of the handle must contain only terminal symbols. For convenience, we refer to the body β as handle rather than $A \to \beta$. Note that we say

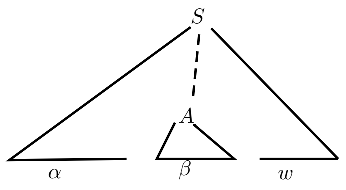

Figure 18.2: A handle $A \to \beta$ in the parse-tree for $\alpha\beta w$

"a handle" instead of "the handle", because grammar could be ambiguous, with more than one right-most derivation of $\alpha\beta w$. However, if a grammar is unambiguous, hen every right sentential form of the grammar has exactly one handle.

The *rightmost derivation in reverse* can be obtained by "handle pruning". That is, we start with a string of terminals $w$ to be parsed. If $w$ is sentence of the grammar at hand, then let $w = \gamma_n$, where γ_n is the nth right-sentential form of some as yet unknown rightmost derivation,

$$S = \gamma_0 \Rightarrow_{rm} \gamma_1 \Rightarrow_{rm} \gamma_2 \Rightarrow_{rm} ... \Rightarrow_{rm} \gamma_{n-1} \Rightarrow_{rm} \gamma_n = w$$

To reconstruct the derivation in reverse order, we locate the handle β_n in γ_n and replace β_n by the head of the relevant production $A_n \to \beta_n$ to obtain the previous right-sentential form γ_{n-1}. We will discuss later as how the handles are to be found.

We then repeat the process, i.e., we locate the handle β_{n-1} in γ_{n-1} and reduce the handle to obtain the right-sentential form γ_{n-2}. Continuing in this way, we obtain the right sentential form consisting only the start symbol, say, S, then we halt and announce successful completion of parsing. The reverse of the sequence of productions used in reductions is a *right most derivation* for the input string.

18.2 Shift-reduce parsing

The shift-reduce parsing is bottom-up parsing where stack holds the grammar symbols and the input buffer holds the rest of the string to be parsed. We will see that handle always appears at top of the stack just before it is identified as the handle. As discussed earlier, \$ will indicate the bottom of the stack, as well right end-marker of the input. Conventionally, when indicating the stack, the top of the stack is its right end. Initially, the stack is empty and the string $w$ is on the input, as shown below:

STACK	*INPUT*
\$	$w$ \$

During the left-to-right scan of input string, the parser shifts zero or more input symbols onto the stack, until it is ready to reduce a string β of grammar symbols on the top of the stack. It then reduces β to the appropriate head of the production symbol. The parser repeats this cycle until it has detected an error or until the stack contents reduces to start symbol of grammar, e.g., S, and simultaneously, the input becomes empty, indicated by \$ symbol, ass shown below.

$STACK$	$INPUT$
$\$\ S$	$\$$

When the parser enters into this configuration, it halts and announces successful completion of parsing. The Table 18.2 shows the steps through which the shift-reduce parser will go through while it is parsing the input $id_1{+}id_2$. The parsing is carried out using the expression grammar shown in Table 18.1 (page 18-1).

Table 18.2: Configurations of a shift-reduce parser on input $id_1 + id_2$

STACK	INPUT	ACTION
$\$$	$id_1 + id_2\ \$$	shift
$\$\ id_1$	$+\ id_2\ \$$	reduce by $F \to id$
$\$\ F$	$+\ id_2\ \$$	reduce by $T \to F$
$\$\ T$	$+\ id_2\ \$$	reduce $E \to T$
$\$\ E$	$+\ id_2\ \$$	shift
$\$\ E\ +$	$id_2\ \$$	shift
$\$\ T + id_2$	$\$$	reduce by $F \to id$
$\$\ E + F$	$\$$	reduce by $T \to F$
$\$\ E + T$	$\$$	reduce by $E \to E + T$
$\$\ E$	$\$$	accept

We have noted that primary operations of this parser are shift and reduce, but there are actually four possible operations:

1. *Shift.* Shift the next input symbol onto the top of the stack.
2. *Reduce.* The right-end of the string (i.e., body of some production) to be reduced must be at the top of the stack. Locate the left end of the string within the stack and decide as what non-terminal should replace it.
3. *Accept.* Declare the successful completion of the parsing.
4. *Error.* Discover a syntax error and call and error recovery routine.

For the shift-reduce parsing, the handle will always appear at the top of the stack, and never inside it. Consider the following case: Non-terminal A is replaced by βBx, and then rightmost terminal B in the body βBx is replaced by γ. This is shown in Fig. 18.3.

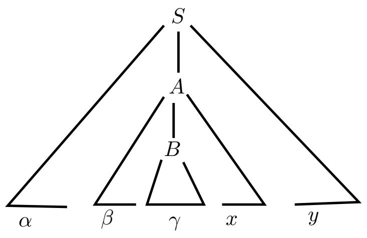

Figure 18.3: Successive steps for rightmost derivation

Let us imagine that the shift-reduce parser has just reached the configuration given below:

$STACK$	$INPUT$
$\$\ \alpha\beta\gamma$	$xy\ \$$

The parser reduces the handle γ to B to reach the configuration:

$\$\ \alpha\beta B$	$xy\ \$$

The parser now shifts the string x onto the stack through a sequence of one or more shifts moves to reach the configurations:

$\$\ \alpha\beta Bx$	$y\ \$$

with handle βBx on top of the stack, and it gets reduced to A, as the Fig. 18.3 shows.

The shift-reduce algorithm is shown as Algorithm 1.

Algorithm 1 Shift-reduce parsing algorithm(Input: sentence w, Stack-empty, output: Parse-tree)

```
1: Push $ onto stack
2: word = nextword()
3: repeat
4:    if (handle of $A \to \beta$ is on stack top) then
5:       pop $|\beta|$ symbols off the stack
6:       push $A$ onto stack
7:    else
8:       if (word $\neq$ eof) then
9:          push word
10:         word = nextword()
11:      else
12:         report syntax error and halt
13:      end if
14:   end if
15: until word=eof & stack contains goal
```

For an input of length n, this shift-reduce parser performs n shifts. It performs a reduction for each step in the derivation, for r steps. It looks for a handle for each iteration of the repeat-until loop, so it must perform $n + r$ handle-finding operations. This is equal to number of nodes in the parse-tree; each shift and each reduce creates a new node in the parse-tree.

18.3 Review Questions

1. In shift-reduce parsing, what do you mean by "derivation in reverse order"? Explain.
2. What is "handle pruning" in such reduce parsing? Explain.

3. What do you understand by "right-mots derivation" and "in reverse" in a shift-reduce parsing? What is the physical position of a handle, when it is about to be reduced?
4. At any moment, what is preferred operation out of "shift" and "reduce"? Justify your answer.
5. What are shift-reduce and reduce-reduce conflicts?

18.4 Exercises

1. For the grammar $S \rightarrow 0\ S\ 1 \mid 0\ 1$, indicate the handle in each of the right-sentential forms:
   (a) 000111
   (b) $00S11$
2. For the grammar $S \rightarrow 0\ S\ 1 \mid 0\ 1$, give the bottom-up parses for the following input strings 000111.
3. For the grammar $S \rightarrow SS+ \mid SS* \mid a$, indicate the handle in each of the following following right-sentential forms:
   (a) $SSS + a * +$
   (b) $SS + a * a+$
   (c) $aaa * a + +$
4. For the grammar $S \rightarrow 0\ S\ 1 \mid 0\ 1$, give the bottom-up parsing for the input string 000111.
5. For the grammar $S \rightarrow SS+ \mid SS* \mid a$, give the bottom-up parsing for the input string $aaa * a + +$.
6. Consider the following grammar where S is the start symbol:

$$S \rightarrow i\ c\ t\ S\ e\ S \mid i\ c\ t\ S \mid a$$

   Compute FOLLOW for each non-terminal of the above grammar.
7. Construct the canonical collection of LR(0) items for the grammar.
8. Use the grammar: $S \rightarrow aSb \mid bSa \mid c$, to parse the following expressions using shift-reduce parsing:
   (a) acb
   (b) bca
   (c) a^2cb^2
   (d) $a^2b^2ca^2b^2$
9. If there are more than one handle available at the same time in a sentential, does it imply that corresponding grammar ambiguous? Justify your answer for Yes/No, and given examples in support or in counter.
10. Use shift-reduce parser to parse the following expressions, using expression grammar:

(a) $id * id + id$

(b) $id * id/id$

(c) $id * id + id * id$

References

[1] Compilers: Principles, Techniques, and Tools (2nd Edition) by Alfred V. Aho, Monica S. Lam, et al., Sep 10, 2006.

[2] Compiler design in C (Prentice-Hall software series) by Allen I Holub, Jan 1, 1990.

[3] Engineering a Compiler, by Keith D. Cooper and Linda Torczon, Morgan Kaufmann Publishers, 2004.

[4] Tools for Large-scale Parser Development, Proceedings of the COLING-2000 Workshop on Efficiency In Large-Scale Parsing Systems, 2000, pp. 54-54, `http://dl.acm.org/citation.cfm?id=2387596.2387604`.

[5] `https://www.antlr.org/`