CS310

Parsing with Context Free Grammars

Today's reference:

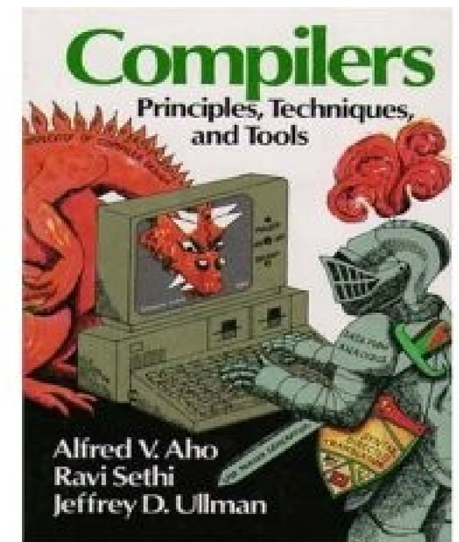
Compilers: Principles, Techniques, and Tools

by: Aho, Sethi, Ullman

aka: The Dragon Book

Section 2.4 page 40

October 20, 2008



Parsing

- Can a string, s, be generated by a grammar?
  - does source code conform to the C grammar?
- For any CFG, we can parse in $O(n^3)$, $n = |s|$
  - O(n) algorithms exist for languages that arise in practice
  - Single left to right scan with one look ahead character
- Top-down vs. Bottom-up
  - describes how you construct the parse tree

Parsing

Example

A -> 0A1

A -> B

B -> #

- Top-down
  - efficient parsers that are more easily constructed by hand
  - We will be concerned with these for now
- Bottom-up
  - handles a larger class of grammars
  - often used in software tools that produce a parser from a grammar

Top Down Parsing

- For some grammars, this can be done with a single left to right scan of the input
  - looking at a single character at a time
  - the *lookahead* character

```
TYPE -> SIMPLE
        | id
        | array [ SIMPLE ] of TYPE
SIMPLE -> integer
        | char
        | num dotdot num                *
```

*from Aho, Sethi, Ullman

Let’s build the parse tree

array [num dotdot num] of integer

Recursive-descent Parsing

- Top down parsing
  - execute a set of recursive procedures to parse
  - one procedure per nonterminal
- Predictive parsing
  - special case of Recursive-descent parsing
  - the lookahead character ***unambiguously*** determines how to choose the next step
    - not all grammars will work

Example

```
procedure type
begin
  if lookahead is in { integer, char, num } then
     simple()
  else if lookahead = id then
     match(id);
  else if lookahead = array then
     match(array); match([); simple; match(]);
     match(of); type;
  else
     error
  endif
end type
```

```
TYPE -> SIMPLE
        | id
        | array [ SIMPLE ] of TYPE
```

Left Recursion

T -> T a x | x

what does this produce?

- Left Recursive Grammar
- Problem?

- Rewrite as *right recursive*

T -> x R

R -> ax R | ε

First

- The lookahead character ***unambiguously*** determines how to choose the next step
- We calculate FIRST(A)

- FIRST(A) is the set of characters that appear as the first symbols of one or more strings generated from A ♦
- For predictive parsing to work without backtracking when A`->`X and A `->`Y exist, FIRST(X) and FIRST(Y) must be disjoint

♦from Aho, Sethi, Ullman

First

- What is FIRST() for each of the nonterminals?

```
TYPE -> SIMPLE
        | id
        | array [ SIMPLE ] of TYPE
SIMPLE -> integer
        | char
        | num dotdot num                *
```

*from Aho, Sethi, Ullman

Simple Parse Table

- Instead of a function, we can build a table to tell us how to parse.

(1) **S -> Q**
(2) **S -> (S)**
(3) **Q -> 1**
(4) **Q -> 0**

	(	)	0	1
S	2	-	1	1
Q	-	-	4	3

Parse Error!