Lecture 15: More Parsing & Semantics

CSCI 131
Fall, 2011

Kim Bruce

Rewrite Grammar

```
       <exp> ::= <term> <termTail>                (1)
  <termTail> ::= <addop> <term> <termTail>        (2)
               | ε                                (3)
      <term> ::= <factor> <factorTail>            (4)
<factorTail> ::= <mulop> <factor> <factorTail>    (5)
               | ε                                (6)
    <factor> ::= ( <exp> )                        (7)
               | NUM                              (8)
               | ID                               (9)
     <addop> ::= + | -                           (10)
     <mulop> ::= * | /                           (11)
```

No left recursion

How do we know which production to take?

Parse Table for Arithmetic

Non-terminals	*ID*	*NUM*	*Addop*	*Mulop*	*(*	*)*	*EOF*
<exp>	1	1			1		
<termTail>			2			3	3
<term>	4	4			4		
<factTail>			6	5		6	6
<factor>	9	8			7		
<addop>			10				
<mulop>				11			

Alternatives to Recursive Descent Parsers

Table-Driven Stack-based Parser

- http://en.wikipedia.org/wiki/LL_parser
- Start with S $ on stack and input $ to be recognized.
- Use table to replace non-terminals on top of stack.
- If terminal on top of stack matches next input then erase both and proceed.
- Success if end up clearing stack and input
- Show with ID * (NUM + NUM)$

Another alternative

- LR(1) parsers -- bottom up, gives right-most derivation.
- YACC is LR(1). ANTLR is LL(1).
- k in LL(k) and LR(k) indicates how many letters of look ahead are necessary -- e.g. length of strings in columns of table.
- Compiler writers are happiest with k=1 to avoid exponential blow-up of table. May have to rewrite grammars.

Scope -- a bit out of order ...

Scope

- Range of instructions where identifier is known
- Static: Scope associated with static text of program.
- Dynamic: Scope associated with execution path of program.

Hole in Scope (Static)

```
program ...
    var M: integer;
    ....
    procedure A ...
        var M: array [1..10] of real;
        begin
            ...
        end;
begin
    ...
end.
```

Static vs Dynamic Scope

```
program ...
    var A : integer;

    procedure Y(B: integer);
        begin
            ...;
            B := A + B;
            ...
        end; {Y}
```

which A?

```
procedure Z(...);
    var A: integer;
    begin
        ...;
        Y(...);
        ...
    end; {Z}
begin {main}
    ...;
    Z(...);
    ...
end.
```

Symbol Table: Built compile-time or run-time?

Semantics

- Operational
- Axiomatic
- Denotational (*Mitchell text.*)

Natural (*Operational*) Semantics

- Arithmetic expressions example on web page
  - ArithSemantics.hs
- How to interpret identifiers?
- Environment: Association list of id's & values.
- Semantics defined recursively on abstract syntax trees.

PCF

- Programming language for Computable Functions
- Includes recursive definitions
- Call-by-value (eager) semantics
- Function application as substitution
- Rewriting semantics

PCF Syntax & Semantics

```
e ::= x | n | true | false | succ | pred | iszero |
      if e then e else e | (fn x => e) | (e e) |
      rec x => e | let x = e1 in e2 end
```

(1) n ⇒ n *for n an integer.*

(2) true ⇒ true, false ⇒ false

(3) error ⇒ error

(4) succ ⇒ succ, *and similarly for the other initial functions*

$$(5)\ \frac{b \Rightarrow \text{true} \qquad e1 \Rightarrow v}{\text{if } b \text{ then } e1 \text{ else } e2 \Rightarrow v}$$

More PCF Semantics

$$(6)\ \frac{b \Rightarrow \text{false} \qquad e2 \Rightarrow v}{\text{if } b \text{ then } e1 \text{ else } e2 \Rightarrow v}$$

$$(7)\ \frac{e1 \Rightarrow \text{succ} \qquad e2 \Rightarrow n}{(e1\ e2) \Rightarrow (n+1)}$$

$$(8)\ \frac{e1 \Rightarrow \text{pred} \quad e2 \Rightarrow 0}{(e1\ e2) \Rightarrow 0} \qquad \frac{e1 \Rightarrow \text{pred} \quad e2 \Rightarrow (n+1)}{(e1\ e2) \Rightarrow n}$$

$$(9)\ \frac{e1 \Rightarrow \text{iszero} \quad e2 \Rightarrow 0}{(e1\ e2) \Rightarrow \text{true}} \qquad \frac{e1 \Rightarrow \text{iszero} \quad e2 \Rightarrow (n+1)}{(e1\ e2) \Rightarrow \text{false}}$$

More PCF Semantics

$$(10)\ \ (\text{fn } x \Rightarrow e) \Rightarrow (\text{fn } x \Rightarrow e)$$

$$(11)\ \frac{e1 \Rightarrow (\text{fn } x \Rightarrow e3) \qquad e2 \Rightarrow v1 \qquad e3[x:=v1] \Rightarrow v}{(e1\ e2) \Rightarrow v}$$

$$(12)\ \frac{e[x:=\text{rec } x \Rightarrow e] \Rightarrow v}{(\text{rec } x \Rightarrow e) \Rightarrow v}$$