LL(1) Parsing

Motivation

- We usually need a parse tree
- Now we have all the machinery we need to produce it!

Parsing

- Create a stack
- This keeps track of the symbols that we have yet to match up with input tokens
- Initially, push start symbol to stack
- Note: We assume we have pseudotoken “$” appended to the input token list

Parse

- Now we cycle through a loop:

```
while input remains:
    let t = symbol of next token of input
    let s = symbol at top of stack
    if s is a terminal:
        if s == t:
            consume next token
            pop stack
    else:
        pop stack
        push symbols in table[s][t] in reverse order
```

Parse

- Problem: This algorithm has no error checking
- We probably want to check for a few error conditions...

Errors

- If we come into the while-loop and stack is empty: Error
  - Indicates we have trailing garbage at end of input

Errors

- If we exit the while-loop and stack is not empty: Error
  - Indicates we had early EOF

Errors

- If s is a terminal but $s \neq t$: Error
  - We expected to see symbol s, but we actually saw symbol t

Errors

- If s is a nonterminal and table[s][t] is empty: Error
  - Symbol s cannot expand into anything that begins with t (or, if s can go to λ, nothing in follow of s leads off with t)

Grammar

- The grammar:

IF → if\b
ELSE → else\b
SEMI → ;
NUM → \d+
ID → \w+
EQ → =
LP → \(
RP → \)
ADDOP → \+
MULOP → *

LBR → \{
RBR → \}

S → stmt SEMI S | λ
a-o-f → ID a-o-f'
a-o-f' → EQ e | LP e RP
cond → IF LP e RP LBR S RBR cond'
cond' → λ | ELSE

LBR S RBR
e → t e'
e' → ADDOP t e' | λ
f → ID | NUM | LP e RP
stmt → a-o-f | cond
t → f t'
t' → MULOP f t' | λ

Table

	$	ADDOP	ELSE	EQ	ID	IF	LP	MULOP	NUM	RBR	RP	SEMI
S	λ	•	•	•	stmt SEMI S	stmt SEMI S	•	•	•	λ	•	•
a-o-f	•	•	•	•	ID a-o-f'	•	•	•	•	•	•	•
a-o-f'	•	•	•	EQ e	•	•	LP e RP	•	•	•	•	•
cond	•	•	•	•	•	IF LP e RP LBR S RBR cond'	•	•	•	•	•	•
cond'	•	•	ELSE LBR S RBR	•	•	•	•	•	•	•	•	λ
e	•	•	•	•	t e'	•	t e'	•	t e'	•	•	•
e'	•	ADDOP t e'	•	•	•	•	•	•	•	•	λ	λ
f	•	•	•	•	ID	•	LP e RP	•	NUM	•	•	•
stmt	•	•	•	•	a-o-f	cond	•	•	•	•	•	•
t	•	•	•	•	f t'	•	f t'	•	f t'	•	•	•
t'	•	λ	•	•	•	•	•	MULOP f t'	•	•	λ	λ

Example

- Input: x=7*4;
- Tokens: ID, EQ, NUM, MULOP, NUM
- Trace out parse actions (in class)

Tree

- How can we get the tree?
- Suppose we have:

```
class TreeNode:
    def __init__(self,sym):
        self.children=[]
        self.sym = sym
        self.token = None
```

- Instead of storing strings to stack, put tree nodes on stack

Loop

```
while input remains:
    let t = symbol of next token of input
    let s = stack.top().sym
    if s is a terminal:
        if s == t:
            stack.top().token = next token of input
            consume token
            pop stack
    else:
        n = stack.top()
        pop stack
        for sym in reversed(table[s][t]):
            c = TreeNode(sym)
            n.children.prepend(c)
            push c to stack
```

Bootstrap

- To begin:
- root = TreeNode(start_symbol)
- stk.push(root)
- When done: “root” is the root of the tree

Example

- Trace previous example but this time, create tree

Assignment

- Write a program which works with the test harness to produce a correct parse tree.

Sources

- Aho, Lam, Sethi, Ullman. Compilers: Principles, Techniques, & Tools (2nd ed).
- K. Louden. Compiler Construction: Principles and Practice.

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org