8 Shift-Reduce Parsing

Wednesday, February 21, 2024 12:07 PM

Grammars :- for specification.
context Free.

What about recognition? $w \in L$

- "The Parsing Problem"
  "recognize a language specified by a c.f. Grammar"

  Algorithm to - build a derivation
  - construct a parse tree.

- TYPES OF PARSERS:
  - Top-Down Parser.
    - construct tree from root (start symbol) to leafs. (w)

    e.g. • "Recursive Descent Parser"
  - Bottom-up Parser.
    - constructs parse tree from leaves to root.

    e.g. "Shift-Reduce Parser"

- THE SHIFT-REDUCE PARSER D. Knuth.

  - Read input from Left to Right
  - Produces Rightmost Derivation

  Called LR-Parser.
  classification: LR(k) k is a number.
  k is the number of symbols from the input the algorithm needs to know to proceed.

  - LR(1)
  - Not a General Parser
    Limited to "LR-Grammars"

- Intuition:

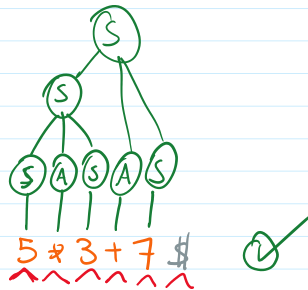
  In a loop, do one of two things
  - Shift :- read next symbol from input
  - Reduce :- build an intermediate node of the parse tree

    i.e. take nods that correspond to a body of a rule, an connect them to a "parent" that correspond

i.e. take nods that correspond to a body of a rule, an connect them to a "parent" that correspond to the head

E.G.

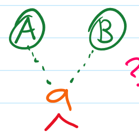
S → SAS | 5 | 3 | 7
A → + | *



- CONFLICTS IN SHIFT-REDUCE PARSER
  - Reduce-Reduce Conflict

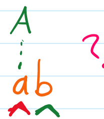
    S → A | B
    A → a
    B → a

    Derivation.
    S
    A
    a



  - Shift-Reduce Conflict

    S → ab | Ab
    A → a



- SHIFT-REDUCE TRACE
  - The algorithm constructs tables from the grammar.
  - These tables drive a "push-down" automata.

```
PROCEDURE shift-reduce ()
  stack.push(0)    // 0 is the start state
  input := w$      // w is the input string
  x := first symbol in w

  WHILE ~stop DO
    s := top of stack
    IF action[s,x] = shift t
      stack.push(t)
      x := next input symbol

    ELSIF action[s,x] = reduce t  // A → β
      pop len(β) symbols from stack
      t := stack.top()
      stack.push( goto[t,A] )
      output( A → β )

    ELSIF action[s,x] = accept
      stop := true
```

E.G.
1) S → CC
2) C → aC
3) C → b

ACTION

	a	b	$
0	S3	S4	
1			acc
2	S6	S7	
3	S3	S4	
4	R3	R3	
5			R1
6	S6	S7	
7			R3
8	R2	R2	
9			R2

GOTO

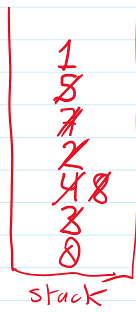
	S	C
0	1	2
1		
2		5
3		8
4		
5		
6		9
7		
8		
9		

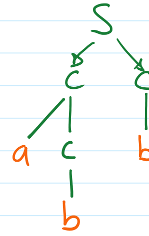
word = abb$

• C → b

stop := true

word = abb$





- XKCD

(AN UNMATCHED LEFT PARENTHESIS CREATES AN UNRESOLVED TENSION THAT WILL STAY WITH YOU ALL DAY.

EOF.