4. CFG 2

Wednesday, February 24, 2021 1:40 PM

Shift reduce parser

- bottom-up
- L-R → rightmost derivation
  - ↓ left to right scanning

* uses stack
* buffer that holds input string
* $ denotes bottom of stack and end of input buffer

Parsing Algorithm

* holds rules for parsing: parsing table
* parses sees input symbol in buffer and

a → shifts: shift input symbol to the stack

b → reduces: takes the handle from the stack and reduces it w/ the help of a production.

input buffer holds tokens

Stack
$aXYZ
$aA

$A \to XYZ$ → reduction

Example

parser gives parsing rule through parsing table

$w = id_1 * id_2$

Stack	input	action
$	$id_1 * id_2$ $	shift
$id_1	$* id_2$ $	reduce F→id
$F	$* id_2$ $	reduce T→F
$T	$* id_2$ $	shift
$T*	id_2 $	shift
$T* id_2	$	reduce F→id
$T* F	$	reduce T→T*F
$T	$	reduce E→T
$E	$	accept

c → Error situations
cannot decide shift or reduce

d → accept

Stack	input
$S	$

→ accept State

$E \to E*T$ $F \to id$
$E \to T$ $F \to (E)$
$T \to T*F$ $T \to F$

→ CFG

how does the parser make parsing decisions

LR(K) Parsing Algorithm

number of symbols used for lookahead

→ Simple L-R parses (SLR) [shift/reduce bottom up]

* maintain a set of states
  - number of L-R(0) items

L-R(0) item of a Grammar G is a production of the grammar with a •

LR(0) item of a Grammar G
is a production of the grammar with a • at any position in the body of the production

$A \to XYZ$
$A \to \cdot XYZ$
$A \to X \cdot YZ$
$A \to XY \cdot Z$
$A \to XYZ \cdot$

• represents how much of a production we have seen

$A \to \cdot XYZ$ // not seen this production
$A \to XYZ \cdot$ // seen this production entirely

Collection of states
canonical LR(0) collection
↳ construct an LR automata (Deterministic) that helps the parser make decisions

1- Augmented Grammar
2- Closure function: closure(I) // generates LR(0) items
3- Go to function: goto(I, X) // defines transitions
} functions

Augmented Grammar G'
→ grammar G with an additional production
$S' \to S$
* parser needs to have a specific production to announce that the string is in an accepting state

Example:
$E \to E+T \mid T$
$T \to T+F \mid F$
$F \to (E) \mid id$

Augmented Grammar:
$E' \to E$
$E \to E+T \mid T$
$T \to T+F \mid F$
$F \to (E) \mid id$

Closure

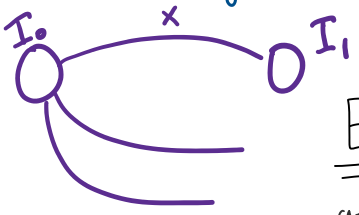
→ Assume I is a set of LR(0) items then the closure of I is formed by using the two items:
1- add all items of I to closure(I)
2- if you have a production of the form
$A \to \alpha . B\beta$ in I
and $B \to \gamma$
$B \to . \gamma$ this item also must be added to closure of I

Example:
LR(0) item: $E' \to .E$
Closure $(E' \to .E)$ = $E \to .E$ (initial state I_0)
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .id$
$F \to .(E)$

Goto (X, I)

- defines transitions
- move the symbol •

in I_0
$A \to \alpha . X\beta$
goto $(X, I_0) \Rightarrow$ closure $(A \to \alpha X . \beta)$



Example

goto (I, x)
goto (I_0, E)
goto (I_0, T)
goto (I_0, F)
goto (I_0, id)
goto $(I_0, '(')$

$E' \to E$
$E \to E. + T$ → cannot be further derived

Building the LR(0) automaton

I. closure $(E' \to E)$

Building the LR(0) automaton

Augmented Grammar

$E' \to E$
$E \to E+T$
$E \to T$
$T \to T*F$
$T \to F$
$F \to id \mid (E)$

I_0 closure $(E' \to E)$

$E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .id$
$F \to .(E)$

I_1 closure (I_0, E)

$E' \to E.$
$E \to E.+T$

I_2 closure (I_0, T)

$E \to T.$
$T \to T.*F$

I_3 closure (I_0, F)

$T \to F.$

I_4 closure (I_0, id)

$F \to id.$

I_5 closure $(I_0, '(')$

$F \to (.E)$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .id$
$F \to .(E)$