Compiler Design

Parse Table Construction

Amey Karkare
Department of Computer Science and Engineering
IIT Kanpur
karkare@iitk.ac.in

Constructing parse table

Augment the grammar

- G is a grammar with start symbol S
- The augmented grammar G' for G has a new start symbol S' and an additional production S' → S
- When the parser reduces by this rule it will stop with accept

2

Production to Use for Reduction

- How do we know which production to apply in a given configuration
- We can guess!
  - May require backtracking
- Keep track of "ALL" possible rules that can apply at a given point in the input string
  - But in general, there is no upper bound on the length of the input string
  - Is there a bound on the number of applicable rules?

Some hands on!

1. E'→ E
2. E→ E + T
3. E→ T
4. T→ T * F
5. T→ F
6. F→ (E)
7. F→ id

Strings to Parse

- id + id + id + id
- id * id * id * id
- id * id + id * id
- id * (id + id) * id

Parser states

- Goal is to know the valid reductions at any given point
- Summarize all possible stack prefixes α as a parser state
- Parser state is defined by a DFA state that reads in the stack α
- Accept states of DFA are unique reductions

Viable prefixes

- α is a viable prefix of the grammar if
  - $\exists w$ such that αw is a right sentential form
  - $\langle\alpha,w\rangle$ is a configuration of the parser
- As long as the parser has viable prefixes on the stack no parser error has been seen
- The set of viable prefixes is a regular language
- We can construct an automaton that accepts viable prefixes

LR(0) items

- An LR(0) item of a grammar G is a production of G with a special symbol "." at some position of the right side
- Thus production $A \rightarrow XYZ$ gives four LR(0) items

  $A \rightarrow .XYZ$

  $A \rightarrow X.YZ$

  $A \rightarrow XY.Z$

  $A \rightarrow XYZ.$

LR(0) items

- An item indicates how much of a production has been seen at a point in the process of parsing
  - Symbols on the left of "." are already on the stacks
  - Symbols on the right of "." are expected in the input

Start state

- Start state of DFA is an empty stack corresponding to S'→.S item
- This means no input has been seen
- The parser expects to see a string derived from S

9

Closure of a state

- **Closure** of a state adds items for all productions whose LHS occurs in an item in the state, just after "."
  - Set of possible productions to be reduced next
  - Added items have "." located at the beginning
  - No symbol of these items is on the stack as yet

10

Example

For the grammar

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

If I is { E' → .E } then closure(I) is

E' → .E
E → .E + T
E → .T
T → .T * F
T → .F
F → .id
F → .(E)

11

Closure operation

- Let I be a set of items for a grammar G
- closure(I) is a set constructed as follows:
  - Every item in I is in closure (I)
  - If A → α.Bβ is in closure(I) and B → γ is a production then B → .γ is in closure(I)
- Intuitively A →α.Bβ indicates that we expect a string derivable from Bβ in input
- If B → γ is a production then we might see a string derivable from γ at this point

12

Goto operation

- Goto(I,X) , where I is a set of items and X is a grammar symbol,
  - is closure of set of item $A \rightarrow \alpha X.\beta$
  - such that $A \rightarrow \alpha.X\beta$ is in I
- Intuitively if I is a set of items for some valid prefix α then goto(I,X) is set of valid items for prefix αX

13

Goto operation

If I is { $E' \rightarrow E.$, $E \rightarrow E. + T$ } then goto(I,+) is

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

14

Sets of items

C : Collection of sets of LR(0) items for grammar G'

C = { closure ({ $S' \rightarrow .S$ }) }

repeat

    for each set of items I in C

        for each grammar symbol X

            if goto (I,X) is not empty and not in C

                ADD goto(I,X) to C

until no more additions to C

15

Example

Grammar:
$E' \rightarrow E$
$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id$

I_0: closure($E' \rightarrow .E$)
$E' \rightarrow .E$
$E \rightarrow .E + T$
$E \rightarrow .T$
$T \rightarrow .T * F$
$T \rightarrow .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I_1: goto(I_0,E)
$E' \rightarrow E.$
$E \rightarrow E. + T$

I_2: goto(I_0,T)
$E \rightarrow T.$
$T \rightarrow T. *F$

I_3: goto(I_0,F)
$T \rightarrow F.$

I_4: goto(I_0,( )
$F \rightarrow (.E)$
$E \rightarrow .E + T$
$E \rightarrow .T$
$T \rightarrow .T * F$
$T \rightarrow .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I_5: goto(I_0,id)
$F \rightarrow id.$

16

I_6: goto(I_1,+)
E → E + .T
T → .T * F
T → .F
F → .(E)
F → .id

I_7: goto(I_2,*)
T → T * .F
F →.(E)
F → .id

I_8: goto(I_4,E)
F → (E.)
E → E. + T

goto(I_4,T) is I_2
goto(I_4,F) is I_3
goto(I_4,( ) is I_4
goto(I_4,id) is I_5

I_9: goto(I_6,T)
E → E + T.
T → T. * F

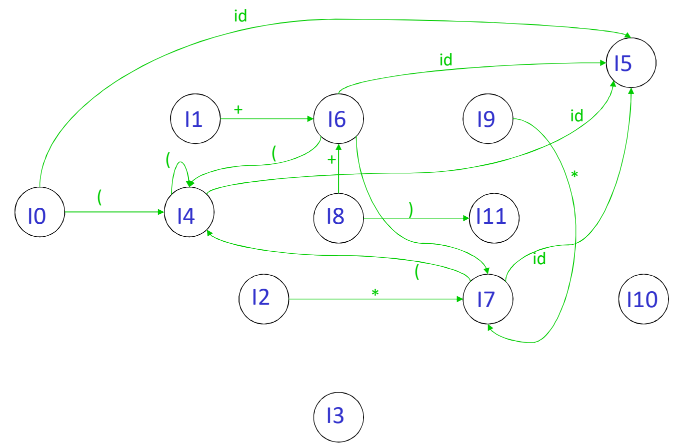
goto(I_6,F) is I_3
goto(I_6,( ) is I_4
goto(I_6,id) is I_5

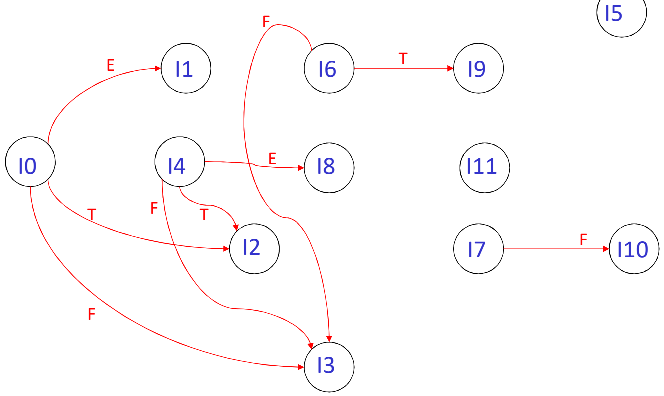
I_{10}: goto(I_7,F)
T → T * F.

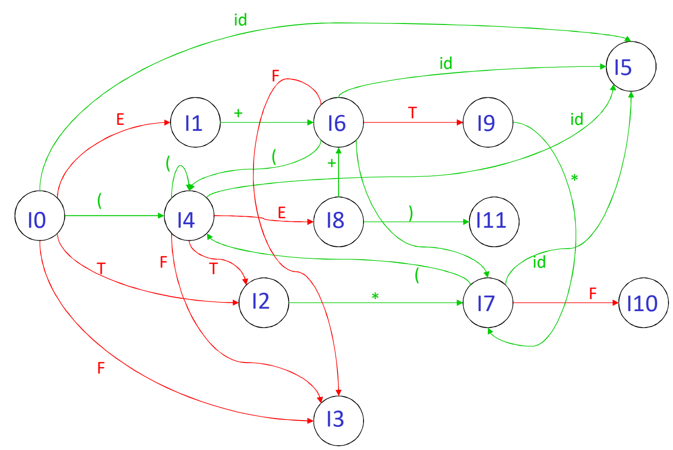
goto(I_7,( ) is I_4
goto(I_7,id) is I_5

I_{11}: goto(I_8,))
F → (E).

goto(I_8,+) is I_6
goto(I_9,*) is I_7







LR(0) (?) Parse Table

- The information is still not sufficient to help us resolve shift-reduce conflict. For example the state:

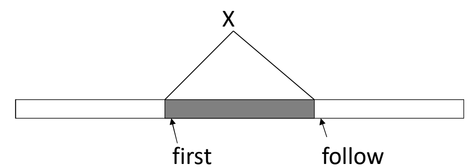
  I_1: E′ → E.

  E → E. + T

- We need some more information to make decisions.

Constructing parse table

- First(α) for a string of terminals and non terminals α is
  - Set of symbols that might begin the fully expanded (made of only tokens) version of α
- Follow(X) for a non terminal X is
  - set of symbols that might follow the derivation of X in the input stream



Compute first sets

- If X is a terminal symbol then first(X) = {X}
- If X → Є is a production then Є is in first(X)
- If X is a non terminal and X → $Y_l Y_2 \ldots Y_k$ is a production, then

  if for some i, a is in first(Y_i)

  and Є is in all of first(Y_j) (such that j<i)

  then a is in first(X)
- If Є is in first (Y_1) … first(Y_k) then Є is in first(X)
- Now generalize to a string α of terminals and non-terminals

Example

- For the expression grammar

  E → T E' E' → +T E' | Є

  T → F T' T' → * F T' | Є

  F → (E) | id

  First(E) = First(T) = First(F)

  = { (, id }

  First(E')

  = {+, Є}

  First(T')

  = { *, Є}

Compute follow sets

1. Place \$ in follow(S) // S is the start symbol
2. If there is a production $A \rightarrow \alpha B \beta$
   then everything in first(β) (except ε) is in follow(B)
3. If there is a production $A \rightarrow \alpha B \beta$ and first(β) contains ε
   then everything in follow(A) is in follow(B)
4. If there is a production $A \rightarrow \alpha B$
   then everything in follow(A) is in follow(B)

Last two steps have to be repeated until the follow sets converge.

25

Example

- For the expression grammar

  $E \rightarrow T\ E'$

  $E' \rightarrow +\ T\ E' \mid \epsilon$

  $T \rightarrow F\ T'$

  $T' \rightarrow *\ F\ T' \mid \epsilon$

  $F \rightarrow (\ E\ ) \mid id$

follow(E) = follow(E') = ?
follow(T) = follow(T') = ?
follow(F) = ?

26

Construct SLR parse table

- Construct $C=\{I_0, \ldots, I_n\}$ the collection of sets of LR(0) items
- If $A \rightarrow \alpha.a\beta$ is in I_i and $goto(I_i,a) = I_j$
  then action[i,a] = shift j
- If $A \rightarrow \alpha.$ is in I_i
  then action[i,a] = reduce $A \rightarrow \alpha$ for all a in follow(A)
- If $S' \rightarrow S.$ is in I_i then action[i,\$] = accept
- If $goto(I_i,A) = I_j$
  then goto[i,A]=j for all non terminals A
- All entries not defined are errors

27

Practice Assignment

Construct SLR parse table for following grammar

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (\ E\ ) \mid digit$

Show steps in parsing of string
9*5+(2+3*7)

- Steps to be followed
  - Augment the grammar
  - Construct set of LR(0) items
  - Construct the parse table
  - Show states of parser as the given string is parsed

28

Notes

- This method of parsing is called SLR (Simple LR)
- LR parsers accept LR(k) languages
  - L stands for left to right scan of input
  - R stands for rightmost derivation
  - k stands for number of lookahead token
- SLR is the simplest of the LR parsing methods. SLR is too weak to handle most languages!
- If an SLR parse table for a grammar does not have multiple entries in any cell then the grammar is unambiguous
- All SLR grammars are unambiguous
- Are all unambiguous grammars in SLR?

29

Does Conflict => Ambiguity?

Example

- Consider following grammar and its SLR parse table:

S' → S
S → L = R
S → R
L → *R
L → id
R → L

I_0: S' → .S
S → .L=R
S → .R
L → .*R
L → .id
R → .L

I_1: goto(I_0, S)
S' → S.

I_2: goto(I_0, L)
S → L.=R
R → L.

Assignment (not to be submitted): Construct rest of the items and the parse table.

31

SLR parse table for the grammar

	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				acc			
2	s6,r6			r6			
3				r3			
4		s4	s5			8	7
5	r5			r5			
6		s4	s5			8	9
7	r4			r4			
8	r6			r6			
9				r2			

The table has multiple entries in action[2,=]

32

- There is both a shift and a reduce entry in action[2,=]. Therefore state 2 has a shift-reduce conflict on symbol "=", However, the grammar is not ambiguous.
- Parse id=id assuming reduce action is taken in [2,=]

Stack	**input**	**action**
0	id=id	shift 5
0 id 5	=id	reduce by L→id
0 L 2	=id	reduce by R→L
0 R 3	=id	error

33

- if shift action is taken in [2,=]

Stack	**input**	**action**
0	id=id$	shift 5
0 id 5	=id$	reduce by L→id
0 L 2	=id$	shift 6
0 L 2 = 6	id$	shift 5
0 L 2 = 6 id 5	$	reduce by L→id
0 L 2 = 6 L 8	$	reduce by R→L
0 L 2 = 6 R 9	$	reduce by S→L=R
0 S 1	$	ACCEPT

34

Another look at the grammar

S' → S
S → L = R
S → R
L → *R
L → id
R → L

- No sentential form of this grammar can start with R=…
- However, the reduce action in action[2,=] generates a sentential form starting with R=
- Therefore, the reduce action is incorrect

Problems in SLR parsing

- In SLR parsing method state i calls for reduction on symbol "a", by rule A→α if I_i contains [A→α.] and "a" is in follow(A)
- However, when state I appears on the top of the stack, the viable prefix βα on the stack may be such that βA can not be followed by symbol "a" in any right sentential form
- Thus, the reduction by the rule A→α on symbol "a" is invalid
- SLR parsers cannot remember the left context

36