Lecture 12

Syntax Analysis

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

February 14, 2025

Take aways from the last class

- Applying symbols to state

Take aways from the last class

- Applying symbols to state
- Goto Operations

Take aways from the last class

- Applying symbols to state
- Goto Operations
- Sets of Item

Take aways from the last class

- Applying symbols to state
- Goto Operations
- Sets of Item
- Limitations of SLR Parser

Take aways from the last class

- Applying symbols to state
- Goto Operations
- Sets of Item
- Limitations of SLR Parser
- Closure operation on LR(1)

Example

- Consider the following grammar
  $S' \rightarrow S$
  $S \rightarrow CC$
  $C \rightarrow cC|d$
- Compute closure(I) where $I = [S' \rightarrow .S, \quad \$]$
  $S \rightarrow .S, \quad \$$
  $S \rightarrow .CC, \quad \$$
  $C \rightarrow .cC, \quad c$
  $C \rightarrow .cC, \quad d$
  $C \rightarrow .d, \quad c$
  $C \rightarrow .d, \quad d$

Example

Construct sets of LR(1) items for the grammar on previous slide

- $I_0 : S \rightarrow .S, \$$
  $S \rightarrow .CC, \$$
  $C \rightarrow .cC$, c/d
  $C \rightarrow .d$, c/d
- $I_1 : goto(I_0, S)$
  $S' \rightarrow S.,\$$
- $I_2 : goto(I_0, C)$
  $S \rightarrow C.C,\$$
  $C \rightarrow .cC,\$$
  $C \rightarrow .d,\$$
- $I_3 : goto(I_0, c)$
  $C \rightarrow c.C$, c/d
  $C \rightarrow .cC$, c/d
  $C \rightarrow .d$, c/d
- $I_4 : goto(I_0, d)$
  $C \rightarrow d.$, c/d
- $I_5 : goto(I_2, C)$
  $S \rightarrow CC.,\$$
- $\mathrm{I}_6 : goto(I_2, c)$
  $C \rightarrow c.C,\$$
  $C \rightarrow .cC,\$$
  $C \rightarrow .d,\$$
- $I_7 : goto(I_2, d)$
  $C \rightarrow d.,\$$
- $I_8 : goto(I_3, C)$
  $C \rightarrow cC.$,c/d
- $I_9 : goto(I_6, C)$
  $C \rightarrow cC.,\$$

Construction of Canonical LR parse table

- Construct $C = I_0, \cdots, I_n$ the sets of LR(1) items

Construction of Canonical LR parse table

- Construct $C = I_0, \cdots, I_n$ the sets of LR(1) items
- If $[A \rightarrow \alpha.a\beta, \quad b]$ is in I_i and $goto(I_i, a) = I_j$ then `action[i,a]=shift j`

Construction of Canonical LR parse table

- Construct $C = I_0, \cdots, I_n$ the sets of LR(1) items
- If $[A \rightarrow \alpha.a\beta, \quad b]$ is in I_i and $goto(I_i, a) = I_j$ then `action[i,a]=shift j`
- If $[A \rightarrow \alpha., a]$ is in I_i then `action[i,a] reduce` $A \rightarrow \alpha$

Construction of Canonical LR parse table

- Construct $C = I_0, \cdots, I_n$ the sets of LR(1) items
- If $[A \to \alpha.a\beta, \quad b]$ is in I_i and $goto(I_i, a) = I_j$ then `action[i,a]=shift j`
- If $[A \to \alpha., a]$ is in I_i then `action[i,a] reduce` $A \to \alpha$
- If $[S' \to S., \$]$ is in I_i then `action[i,$] = accept`

Construction of Canonical LR parse table

- Construct $C = I_0, \cdots, I_n$ the sets of LR(1) items
- If $[A \rightarrow \alpha.a\beta, \quad b]$ is in I_i and $goto(I_i, a) = I_j$ then `action[i,a]=shift j`
- If $[A \rightarrow \alpha., a]$ is in I_i then `action[i,a] reduce` $A \rightarrow \alpha$
- 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

Parse table

State	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LALR Parse Table

- Look Ahead LR parsers

LALR Parse Table

- Look Ahead LR parsers
- Consider a pair of similar looking states (same kernel and different lookaheads) in the set of LR(1) items

LALR Parse Table

- Look Ahead LR parsers
- Consider a pair of similar looking states (same kernel and different lookaheads) in the set of LR(1) items

  $I_4 : C \rightarrow d., \quad c/d$

  $I_7 : C \rightarrow d., \quad \$$

LALR Parse Table

- Look Ahead LR parsers
- Consider a pair of similar looking states (same kernel and different lookaheads) in the set of LR(1) items
  $I_4 : C \rightarrow d., \quad c/d$
  $I_7 : C \rightarrow d., \quad \$$
- Replace I_4 and I_7 by a new state I_{47} consisting of $(C \rightarrow d., \quad c/d/\$)$

LALR Parse Table

- Look Ahead LR parsers
- Consider a pair of similar looking states (same kernel and different lookaheads) in the set of LR(1) items
  $I_4 : C \rightarrow d., \quad c/d$
  $I_7 : C \rightarrow d., \quad \$$
- Replace I_4 and I_7 by a new state I_{47} consisting of $(C \rightarrow d., \quad c/d/\$)$
- Similarly I_3 & I_6 and I_8 & I_9 form pairs

LALR Parse Table

due to more number of states in CLR, it is memory inefficient.
Decrease number of states in LALR

- Look Ahead LR parsers
- Consider a pair of similar looking states (same kernel and different lookaheads) in the set of LR(1) items
  $I_4 : C \rightarrow d., \quad c/d$
  $I_7 : C \rightarrow d., \quad \$$
- Replace I_4 and I_7 by a new state I_{47} consisting of $(C \rightarrow d., \quad c/d/\$)$
- Similarly I_3 & I_6 and I_8 & I_9 form pairs
- Merge LR(1) items having the same core

Construct LALR parse table

- Construct $C = I_0, \cdots, I_n$ set of LR(1) items

Construct LALR parse table

- Construct $C = I_0, \cdots, I_n$ set of LR(1) items
- For each core present in LR(1) items find all sets having the same core and replace these sets by their union

Construct LALR parse table

- Construct $C = I_0, \cdots, I_n$ set of LR(1) items
- For each core present in LR(1) items find all sets having the same core and replace these sets by their union
- Let $C' = J_0, \cdots, J_m$ be the resulting set of items

Construct LALR parse table

- Construct $C = I_0, \cdots, I_n$ set of LR(1) items
- For each core present in LR(1) items find all sets having the same core and replace these sets by their union
- Let $C' = J_0, \cdots, J_m$ be the resulting set of items
- Construct action table as was done earlier

Construct LALR parse table

- Construct $C = I_0, \cdots, I_n$ set of LR(1) items
- For each core present in LR(1) items find all sets having the same core and replace these sets by their union
- Let $C' = J_0, \cdots, J_m$ be the resulting set of items
- Construct action table as was done earlier
- Let $J = I_1 \cup I_2 \cdots \cup I_k$
  since $I_1, I_2 \cdots, I_k$ have same core, $goto(J, X)$ will have the same core
  Let $K = goto(I_1, X) \cup goto(I2, X) \cdots goto(I_k, X)$ the $goto(J, X) = K$

LALR parse table ...

State	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Important points

- SLR and LALR parse tables have same number of states.

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
  - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \to \alpha., a], [Y \to \gamma.a\beta, b]$

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
    - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \rightarrow \alpha., a], [Y \rightarrow \gamma.a\beta, b]$
    - Then there must have been a state in the LR parser with the same core

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
  - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \rightarrow \alpha., a], [Y \rightarrow \gamma.a\beta, b]$
  - Then there must have been a state in the LR parser with the same core
  - Contradiction; because LR parser did not have conflicts

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
  - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \to \alpha., a], [Y \to \gamma.a\beta, b]$
  - Then there must have been a state in the LR parser with the same core
  - Contradiction; because LR parser did not have conflicts
- LALR parser can have new reduce-reduce conflicts

Important points

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
  - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \rightarrow \alpha., a], [Y \rightarrow \gamma.a\beta, b]$
  - Then there must have been a state in the LR parser with the same core
  - Contradiction; because LR parser did not have conflicts
- LALR parser can have new reduce-reduce conflicts
  - Assume states $[X \rightarrow \alpha., a], [Y \rightarrow \alpha., b] and [X \rightarrow \alpha., b], [Y \rightarrow \alpha., a]$

Important points

and CLR have more number of states than LALR | SLR...
states (CLR) >= states (LALR) = states (SLR)

- SLR and LALR parse tables have same number of states.
- Merging items never produces shift/reduce conflicts but may produce reduce/reduce conflicts.
- New conflicts can not be of shift reduce kind:
  - Assume there is a shift reduce conflict in some state of LALR parser with items $[X \rightarrow \alpha., a], [Y \rightarrow \gamma.a\beta, b]$
  - Then there must have been a state in the LR parser with the same core
  - Contradiction; because LR parser did not have conflicts

LR means CLR by default.

- LALR parser can have new reduce-reduce conflicts
  - Assume states $[X \rightarrow \alpha., a], [Y \rightarrow \alpha., b] and [X \rightarrow \alpha., b], [Y \rightarrow \alpha., a]$
  - Merging the two states produces $[X \rightarrow \alpha., a/b], [Y \rightarrow \alpha., a/b]$

there are two reduce possibilities when a/b comes in input when this state is at top of stack,

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables
- There are direct, complicated but efficient algorithms to develop LALR parsers

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables
- There are direct, complicated but efficient algorithms to develop LALR parsers
- Relative power of various classes

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables
- There are direct, complicated but efficient algorithms to develop LALR parsers
- Relative power of various classes
  - SLR(1) $\leq$ LALR(1) $\leq$ LR(1)

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables
- There are direct, complicated but efficient algorithms to develop LALR parsers
- Relative power of various classes
  - SLR(1) $\leq$ LALR(1) $\leq$ LR(1)
  - SLR(k) $\leq$ LALR(k) $\leq$ LR(k)

Important points

- Practical LALR parsers are not built by first making canonical LR parse tables
- There are direct, complicated but efficient algorithms to develop LALR parsers
- Relative power of various classes
  - SLR(1) $\leq$ LALR(1) $\leq$ LR(1)
  - SLR(k) $\leq$ LALR(k) $\leq$ LR(k)
  - LL(k) $\leq$ LR(k)

Error Recovery

- An error is detected when an entry in the action table is found to be empty.

Error Recovery

- An error is detected when an entry in the action table is found to be empty.
- Panic mode error recovery can be implemented as follows:

Error Recovery

errors will not arise from goto table

- An error is detected when an entry in the action table is found to be empty.
- Panic mode error recovery can be implemented as follows:
  - scan down the stack until a state S with a goto on a particular nonterminal A is found.

Error Recovery

- An error is detected when an entry in the action table is found to be empty.
- Panic mode error recovery can be implemented as follows:
  - scan down the stack until a state S with a goto on a particular nonterminal A is found.
  - Discard zero or more input symbols until a symbol a is found that can legitimately $follow(A)$.

Error Recovery

- An error is detected when an entry in the action table is found to be empty.
- Panic mode error recovery can be implemented as follows:
  - scan down the stack until a state S with a goto on a particular nonterminal A is found.
  - Discard zero or more input symbols until a symbol a is found that can legitimately $follow(A)$.
  - stack the state $goto[S, A]$ and resume parsing.

Error Recovery

- An error is detected when an entry in the action table is found to be empty.
- Panic mode error recovery can be implemented as follows:
  - scan down the stack until a state S with a goto on a particular nonterminal A is found.
  - Discard zero or more input symbols until a symbol a is found that can legitimately *follow*(A).
  - stack the state $goto[S, A]$ and resume parsing.
- Choice of A: Normally these are non terminals representing major program pieces such as an expression, statement or a block. For example if A is the nonterminal stmt, a might be semicolon or end

Parser Generator

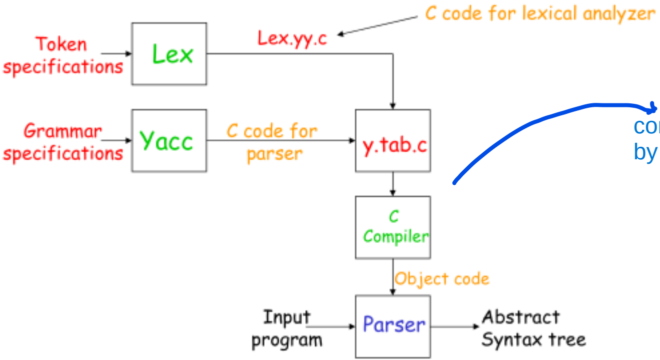
- Some common LR parser generators
  - YACC: Yet Another Compiler Compiler
  - Bison: GNU Software

by default, they generate LALR parsers.

- Yacc/Bison source program specification (accept LALR grammars)

declaration
%%
translation rules
%%
supporting C routines

Yacc and Lex schema



compile both y.tab.c (parser) and lex.yy.c (lexer) by using G++ or GCC.