Lecture 11

Syntax Analysis

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

February 12, 2025

Take aways from the last class

- Parser State

Take aways from the last class

- Parser State
- Agumentation of the Grammar

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items
- Start State

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items
- Start State
- Closure

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items
- Start State
- Closure
- Action

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items
- Start State
- Closure
- Action
- Goto

Take aways from the last class

- Parser State
- Agumentation of the Grammar
- LR(0) items
- Start State
- Closure
- Action
- Goto
- Parse Table creation

Applying symbols in a state

Applying symbols in a state

- Create new state and include all the items that have appropriate input symbol just after the "."

Applying symbols in a state

- Create new state and include all the items that have appropriate input symbol just after the "."
- Advance "." in those items and take closure

Goto operation

Goto operation

- $Goto(I, X)$, where I is a set of items and X is a grammar symbol

Goto operation

- $Goto(I, X)$, where I is a set of items and X is a grammar symbol
  - is closure of set of item $A \rightarrow \alpha X.\beta$

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

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

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
- If I is $E' \rightarrow E.$
  E $\rightarrow E. + T$

  then goto(I,+) is

  $E \rightarrow E + .T$

  $T \rightarrow .T * F$

  $T \rightarrow .F$

  $F \rightarrow .(E)$

  $F \rightarrow .id$

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
        and each grammar symbol X
            such that goto (I,X) is not empty and not in C
                ADD goto(I,X) to C
until no more additions
```

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ the collection of sets of $LR(0)$ items.

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ the collection of sets of $LR(0)$ items.
- If $A \to \alpha.a\beta$ is in I_i and $goto(I_i, a) = I_j$ then `action[i,a] = shift j`

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ 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 $follow(A)$

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ 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`

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ 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

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ 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

Construct SLR(Simple LR) parse table

- Construct $C = I_0, \cdots, I_n$ 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

SLR is too weak to handle most languages!

Homework

- Create SLR Parse table for the following grammar (homework)
  $S' \rightarrow S$
  $S \rightarrow L = R$
  $S \rightarrow R$
  $L \rightarrow *R$
  $L \rightarrow id$
  $R \rightarrow L$

Parse Table

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,=]

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$
- However, the reduce action in action[2,=] generates a sentential form starting with R=$\cdots$

Problems in SLR parsing

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

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$
- However, the reduce action in action[2,=] generates a sentential form starting with R=$\cdots$
- Therefore, the reduce action is incorrect
- In SLR parsing method state i calls for reduction on symbol "a", by rule $A \rightarrow \alpha$ if I_i contains $[A \rightarrow \alpha.]$ and "a" is in follow(A)

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$
- However, the reduce action in action[2,=] generates a sentential form starting with R=$\cdots$
- Therefore, the reduce action is incorrect
- In SLR parsing method state i calls for reduction on symbol "a", by rule $A \rightarrow \alpha$ if I_i contains $[A \rightarrow \alpha.]$ and "a" is in follow(A)
- However, when state I appears on the top of the stack, the viable prefix $\beta\alpha$ on the stack may be such that $\beta\alpha$ can not be followed by symbol "a" in any right sentential form

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$
- However, the reduce action in action[2,=] generates a sentential form starting with R=$\cdots$
- Therefore, the reduce action is incorrect
- In SLR parsing method state i calls for reduction on symbol "a", by rule $A \rightarrow \alpha$ if I_i contains $[A \rightarrow \alpha.]$ and "a" is in follow(A)
- However, when state I appears on the top of the stack, the viable prefix $\beta\alpha$ on the stack may be such that $\beta\alpha$ can not be followed by symbol "a" in any right sentential form
- Thus, the reduction by the rule $A \rightarrow \alpha$ on symbol "a" is invalid

Problems in SLR parsing

- No sentential form of this grammar can start with $R = \cdots$
- However, the reduce action in action[2,=] generates a sentential form starting with R=$\cdots$
- Therefore, the reduce action is incorrect
- In SLR parsing method state i calls for reduction on symbol "a", by rule $A \rightarrow \alpha$ if I_i contains $[A \rightarrow \alpha.]$ and "a" is in follow(A)
- However, when state I appears on the top of the stack, the viable prefix $\beta\alpha$ on the stack may be such that $\beta\alpha$ can not be followed by symbol "a" in any right sentential form
- Thus, the reduction by the rule $A \rightarrow \alpha$ on symbol "a" is invalid
- SLR parsers cannot remember the left context

Canonical LR Parsing

- Carry extra information in the state so that wrong reductions by $A \rightarrow \alpha$ will be ruled out

Canonical LR Parsing

- Carry extra information in the state so that wrong reductions by $A \to \alpha$ will be ruled out
- Redefine LR items to include a terminal symbol as a second component (look ahead symbol).

Canonical LR Parsing

- Carry extra information in the state so that wrong reductions by $A \rightarrow \alpha$ will be ruled out
- Redefine LR items to include a terminal symbol as a second component (look ahead symbol).
- The general form of the item becomes $[A \rightarrow \alpha.\beta, a]$ which is called LR(1) item.

Canonical LR Parsing

- Carry extra information in the state so that wrong reductions by $A \rightarrow \alpha$ will be ruled out
- Redefine LR items to include a terminal symbol as a second component (look ahead symbol).
- The general form of the item becomes $[A \rightarrow \alpha.\beta, a]$ which is called LR(1) item.
- Item $[A \rightarrow \alpha., a]$ calls for reduction only if next input is a. The set of symbols "a"s will be a subset of $Follow(A)$

Closure(I)

```
repeat
    for each item [A → α.Bβ, a] in I
        for each production B → γ in G′
        and for each terminal b in First(βa)
            add item [B → .γ, b] to I
until no more additions to I
```

Example

- Consider the following grammar
  $S' \rightarrow S$
  $S \rightarrow CC$
  $C \rightarrow cC|d$

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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