

LR Parsing

Lecture Notes by
Profs Aiken and Necula (UCB)

CS780(Prasad) L16LR 1

Outline

- Review of SLR parsing
- Limits of SLR parsing
- LR parsing
- LALR parsing
- Implementation of semantic actions
- Using parser generators

CS780(Prasad) L16LR 2

Review of SLR(1) Parsing

- LR parser maintains a stack
 $\langle sym_1, state_1 \rangle \dots \langle sym_n, state_n \rangle$
 $state_n$ is the final state of the DFA on $sym_1 \dots sym_n$
- **Goto table**: the transition function of the DFA
 - $Goto[i,A] = j$ if $state_i \xrightarrow{A} state_j$
- **Action table**: for each state and terminal:

$$Action[i, a] = \begin{cases} \text{Shift } j \\ \text{Reduce } X \rightarrow \alpha \\ \text{Accept} \\ \text{Error} \end{cases}$$

CS780(Prasad) L16LR 3

LR Parsing Algorithm

```

Let I = w$ be initial input
Let j = 0
Let DFA state 1 have item S' -> S
Let stack = ( dummy, 1 )
repeat
  case action[top_state(stack), I[j]] of
    shift k: push ( I[j++], k )
    reduce X -> A:
      pop |A| pairs,
      push (X, Goto[X,top_state(stack)])
    accept: halt normally
    error: halt and report error
  
```

CS780(Prasad) L16LR 4

Review. Items

- An item $[X \rightarrow \alpha.\beta]$ says that
 - the parser is looking for an X
 - it has an α on top of the stack
 - Expects to find a string derived from β next in the input
- Notes:
 - $[X \rightarrow \alpha.a\beta]$ means that a should follow. Then we can shift it and still have a viable prefix.
 - $[X \rightarrow \alpha.]$ means that we could reduce X
 - But this is not always a good idea !

CS780(Prasad) L16LR 5

SLR(1) Action Table

For each state s_i and terminal a

- If s_i has item $X \rightarrow \alpha.a\beta$ and $Goto[i,a] = j$ then
 $Action[i,a] = \text{shift } j$
- If s_i has item $S' \rightarrow S$. then $Action[i,\$] = \text{accept}$
- If s_i has item $X \rightarrow \alpha.$ and $a \in Follow(X)$ and $X \neq S'$ then
 $Action[i,a] = \text{reduce } X \rightarrow \alpha$
- Otherwise, $Action[i,a] = \text{error}$

CS780(Prasad) L16LR 6

Limits of SLR Parsing

- SLR(1) is the simplest LR parsing method
- SLR(1) is almost powerful enough, but ...
- ... some common programming language constructs are not SLR(1).
- Consider the grammar

$$\begin{aligned} S &\rightarrow L = E \mid E \\ L &\rightarrow * E \mid \text{id} \\ E &\rightarrow L \end{aligned}$$

CS780(Prasad)

L16LR

7

Limits of SLR Parsing (cont.)

- Consider two states of the DFA for recognizing viable prefixes

$$\begin{aligned} S' &\rightarrow \cdot S & S &\rightarrow L \cdot = E \\ S &\rightarrow \cdot L = E & \Rightarrow^L & E \rightarrow L \cdot \\ S &\rightarrow \cdot E & & \\ L &\rightarrow \cdot * E & & \\ L &\rightarrow \cdot \text{id} & & \\ E &\rightarrow \cdot L & & \end{aligned}$$

SLR(1) parser on input "="

- shift (item $L \cdot = E$)
- reduce by $E \rightarrow L$ (since "=" $\in \text{Follow}(E)$)

CS780(Prasad)

L16LR

8

What's The Problem?

- The grammar is not SLR(1), but why?
- Focus on the reduce move in the second state
 - We are in the context of $S \rightarrow E \rightarrow L$
 - No = can follow E in this context
 - Even though = $\in \text{Follow}(E)$ (in $S \rightarrow L = E \rightarrow *E = E$)
 - The reduce move should not happen if an = follows in this context.

CS780(Prasad)

L16LR

9

What's The Problem? (Cont.)

- **Problem:** the SLR table has too many reduce actions.
 - Using Follow is too coarse.
- In any given context, only some elements of Follow can actually follow a non-terminal.
- For example:
 - $\text{Follow}(E) = \{=, \$\}$, but
 - In context $S \rightarrow E$ only \$ can follow E
 - In context $S \rightarrow L = E \rightarrow *E_1 = E$ only = can follow E_1

CS780(Prasad)

L16LR

10

One Way to Fix The Problem: LR(1) Items

- **Idea:**
 - refine Follow based on context.
 - The context is described through items.
- An LR(1) item is a pair $[X \rightarrow \alpha \cdot \beta, a]$ where $X \rightarrow \alpha \beta$ is a production and a is the lookahead token or \$
- LR(k) is similar but with k tokens of lookahead
 - In practice, k = 1

CS780(Prasad)

L16LR

11

LR(1) Items. Intuition

- $[X \rightarrow \alpha \cdot \beta, a]$ describes a state of the parser:
 - We are trying to find an X, and
 - We have α already on top of the stack, and
 - We expect to see a prefix derived from βa
- Back to reduce actions: have an $[X \rightarrow \alpha \cdot, a]$
 - Perform the reduce **only** if next token is a !
 - Will have fewer reduce actions
 - Not for all $b \in \text{Follow}(X)$

CS780(Prasad)

L16LR

12

Constructing Sets of LR(1) Items (1)

- Similar to construction for LR(0).
- The states of the NFA are the LR(1) items of G.
- The start state is $[S' \rightarrow \cdot S, \$]$

CS780(Prasad)

L16LR

13

Constructing Sets of LR(1) Items (2)

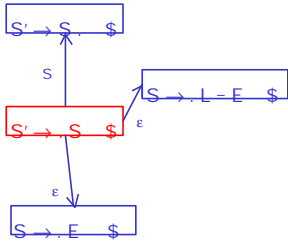
1. For each LR(1) item $[Y \rightarrow \alpha.X\beta, a]$
Add an X-transition
 $[Y \rightarrow \alpha.X\beta, a] \xrightarrow{X} [Y \rightarrow \alpha X\beta, a]$
2. For each LR(1) item $[Y \rightarrow \alpha.X\beta, a]$
For each production $X \rightarrow \gamma$
For each terminal $b \in \text{First}(\beta a)$
Add an ϵ transition
 $[Y \rightarrow \alpha.X\beta, a] \xrightarrow{\epsilon} [X \rightarrow \cdot \gamma, b]$

CS780(Prasad)

L16LR

14

NFA for Viable Prefixes in Detail (1)

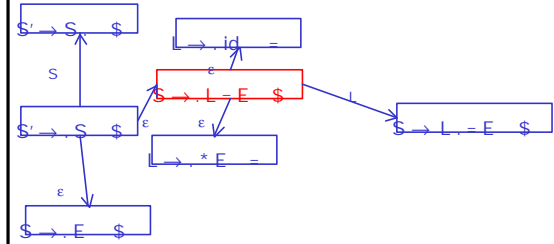


CS780(Prasad)

L16LR

15

NFA for Viable Prefixes in Detail (2)

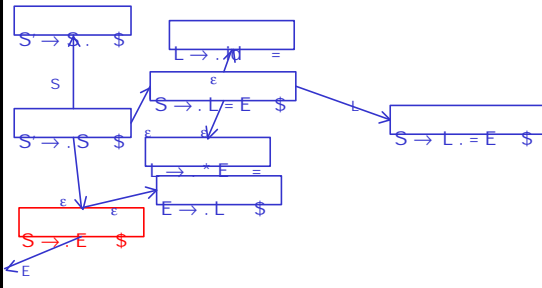


CS780(Prasad)

L16LR

16

NFA for Viable Prefixes in Detail (3)

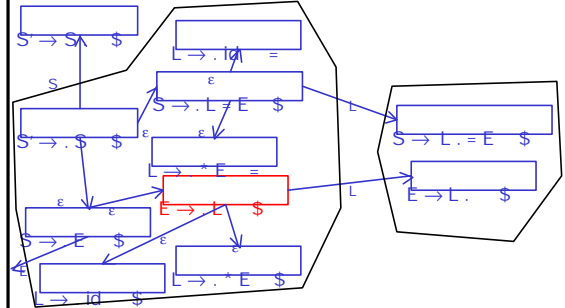


CS780(Prasad)

L16LR

17

NFA for Viable Prefixes in Detail (4)



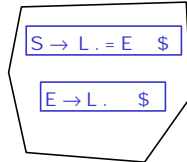
CS780(Prasad)

L16LR

18

An Example Revisited

- Consider the state from last slide



- LR(1) parser on input " $=$ "
 - only shift (item $L \cdot = E$)

CS780(Prasad)

L16LR

19

Constructing LR(1) Parsing Tables

- Add a dummy $S' \rightarrow S$ production
- Construct the NFA of LR(1) items as before
- Convert the NFA into a DFA
- Goto is defined exactly as before:

$$\text{Goto}[i, A] = j \quad \text{if} \quad \text{state}_i \xrightarrow{A} \text{state}_j$$

(the transition function of the DFA)

CS780(Prasad)

L16LR

20

Constructing LR(1) Parsing Tables (Cont.)

- For each state s_i of the DFA and terminal a
 - If s_i has item $[X \rightarrow \alpha \cdot \beta, c]$ and $\text{Goto}[i, a] = j$ then
 $\text{action}[i, a] = \text{shift } j$
 - If s_i has item $[X \rightarrow \alpha \cdot, a]$ and $X \neq S$ then
 $\text{action}[i, a] = \text{reduce } X \rightarrow \alpha$
 - If s_i has item $[S' \rightarrow S \cdot, \$]$ then
 $\text{action}[i, \$] = \text{accept}$
 - Otherwise,
 $\text{action}[i, a] = \text{error}$
- LR(1) grammar \Leftrightarrow $\text{action}[i, a]$ uniquely defined

CS780(Prasad)

L16LR

21

LALR Parsing

- Two bottom-up parsing methods: SLR and LR
- Which one we use? Neither
 - SLR is not powerful enough.
 - LR parsing tables are too big (1000's of states vs. 100's of states for SLR).
- In practice, use **LALR(1)**
 - Stands for **Look-Ahead LR**
 - A compromise between SLR(1) and LR(1)

CS780(Prasad)

L16LR

22

LALR Parsing (Cont.)

- Rough intuition: A LALR(1) parser for G has
 - The number of states of an SLR parser.
 - Some of the lookahead discrimination of LR(1).
- Idea: construct the DFA for the LR(1).
- Then merge the DFA states whose items differ only in the lookahead tokens
 - We say that such states have the same core.

CS780(Prasad)

L16LR

23

The Core of a Set of LR Item

- Definition:** The **core** of a set of LR items is the set of first components.
- Example:** the core of
 $\{ [X \rightarrow \alpha \cdot \beta, b], [Y \rightarrow \gamma \cdot \delta, d] \}$
is
 $\{ X \rightarrow \alpha \cdot \beta, Y \rightarrow \gamma \cdot \delta \}$
- The core of an LR item is an LR(0) item.

CS780(Prasad)

L16LR

24

A LALR(1) DFA

- Repeat until all states have distinct core.
 - Choose two distinct states with same core.
 - Merge the states by creating a new one with the union of all the items.
 - Point edges from predecessors to new state.
 - New state points to all the previous successors.



CS780(Prasad)

L16LR

25

The LALR Parser Can Have Conflicts

- Consider for example the LR(1) states
 - $\{[X \rightarrow \alpha., a], [Y \rightarrow \beta., b]\}$
 - $\{[X \rightarrow \alpha., b], [Y \rightarrow \beta., a]\}$
- And the merged LALR(1) state
 - $\{[X \rightarrow \alpha., a/b], [Y \rightarrow \beta., a/b]\}$
- Has a new reduce-reduce conflict.
- In practice such cases are rare.

CS780(Prasad)

L16LR

26

LALR vs. LR Parsing

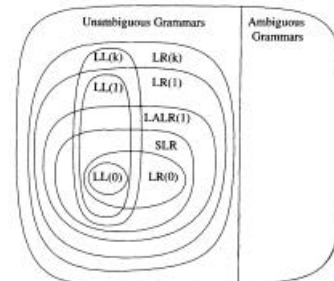
- LALR languages are not natural.
 - They are an efficiency hack on LR languages
- Any reasonable programming language has an LALR(1) grammar.
- LALR(1) has become a standard for programming languages and for parser generators.

CS780(Prasad)

L16LR

27

A Hierarchy of Grammar Classes



CS780(Prasad)

L16LR

28

Semantic Actions

- We can now illustrate how semantic actions are implemented for LR parsing.
- Keep attributes on the stack.
- On **shift** a , push attribute for a on stack.
- On **reduce** $X \rightarrow \alpha$
 - pop attributes for α
 - compute attribute for X
 - and push it on the stack

CS780(Prasad)

L16LR

29

Performing Semantic Actions. Example

- Recall the example from earlier lecture

$$\begin{array}{l}
 E \rightarrow T + E_1 \quad \{ E.val = T.val + E_1.val \} \\
 \quad | T \quad \quad \quad \{ E.val = T.val \} \\
 T \rightarrow int * T_1 \quad \{ T.val = int.val * T_1.val \} \\
 \quad | int \quad \quad \quad \{ T.val = int.val \}
 \end{array}$$

- Consider the parsing of the string $3 * 5 + 8$

CS780(Prasad)

L16LR

30

Performing Semantic Actions. Example

int * int + int	shift
int ₃ * int + int	shift
int ₃ * int + int	shift
int ₃ * int ₅ + int	reduce T → int
int ₃ * T ₅ + int	reduce T → int * T
T ₁₅ + int	shift
T ₁₅ + int	shift
T ₁₅ + int ₈	reduce T → int
T ₁₅ + T ₈	reduce E → T
T ₁₅ + E ₈	reduce E → T + E
E ₂₃	accept

CS780(Prasad)

L16LR

31

Notes

- The previous discussion shows how synthesized attributes are computed by LR parsers.
- It is also possible to compute inherited attributes in an LR parser.

CS780(Prasad)

L16LR

32

Using Parser Generators

- Most common parser generators are LALR(1).
- A parser generator constructs a LALR(1) table.
- And reports an error when a table entry is multiply defined:
 - A shift and a reduce. Called **shift/reduce** conflict
 - Multiple reduces. Called **reduce/reduce** conflict
- An ambiguous grammar will generate conflicts.
- What do we do in that case?

CS780(Prasad)

L16LR

33

Shift/Reduce Conflicts

- Typically due to ambiguities in the grammar.
- **Classic example:** the dangling else
 - $S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid \text{OTHER}$
- Will have DFA state containing
 - $[S \rightarrow \text{if } E \text{ then } S, \quad \text{else}]$
 - $[S \rightarrow \text{if } E \text{ then } S. \text{ else } S, \quad x]$
- if **else** follows, then we can **shift** or **reduce**
- Default (bison, CUP, etc.) is to shift
 - Default behavior is as needed in this case.

CS780(Prasad)

L16LR

34

More Shift/Reduce Conflicts

- Consider the ambiguous grammar
 - $E \rightarrow E + E \mid E * E \mid \text{int}$
- We will have the states containing
 - $[E \rightarrow E * . E, +]$ $[E \rightarrow E * E., +]$
 - $[E \rightarrow . E + E, +] \Rightarrow^E [E \rightarrow E. + E, +]$
 - ...
- Again we have a shift/reduce on input +
 - We need to reduce (* binds more tightly than =)
 - Recall solution: declare the precedence of * and =

CS780(Prasad)

L16LR

35

More Shift/Reduce Conflicts

- In bison, declare **precedence** and **associativity**:
 - `%left +`
 - `%left *`
- Precedence of a rule = that of its last terminal
 - See bison manual for ways to override this default.
- Resolve **shift/reduce** conflict with a **shift** if:
 - no precedence declared for either rule or terminal
 - input terminal has higher precedence than the rule
 - the precedences are the same and right associative

CS780(Prasad)

L16LR

36

Using Precedence to Solve S/R Conflicts

- Back to our example:

$$\begin{array}{l} [E \rightarrow E * . E, +] \quad [E \rightarrow E * E, +] \\ [E \rightarrow . E + E, +] \Rightarrow^E [E \rightarrow E . + E, +] \\ \dots \qquad \qquad \qquad \dots \end{array}$$

- Will choose reduce because precedence of rule $E \rightarrow E * E$ is higher than of terminal $+$

CS780(Prasad)

L16LR

37

Using Precedence to Solve S/R Conflicts

- Same grammar as before

$$E \rightarrow E + E \mid E * E \mid \text{int}$$

- We will also have the states

$$\begin{array}{l} [E \rightarrow E + . E, +] \quad [E \rightarrow E + E, +] \\ [E \rightarrow . E + E, +] \Rightarrow^E [E \rightarrow E . + E, +] \\ \dots \qquad \qquad \qquad \dots \end{array}$$

- Now we also have an **S/R conflict** on input $+$
 - We choose **reduce** because $E \rightarrow E + E$ and $+$ have the same precedence and $+$ is left-associative.

CS780(Prasad)

L16LR

38

Using Precedence to Solve S/R Conflicts

- Back to our dangling else example

$$\begin{array}{l} [S \rightarrow \text{if } E \text{ then } S, \quad \text{else}] \\ [S \rightarrow \text{if } E \text{ then } S. \text{ else } S, \quad x] \end{array}$$

- Can eliminate conflict by declaring **else** with higher precedence than **then**.
- But this starts to look like "hacking the tables".
- Best to avoid overuse of precedence declarations, or you'll end with unexpected parse trees.

CS780(Prasad)

L16LR

39

Reduce/Reduce Conflicts

- Usually due to gross ambiguity in the grammar

- Example:** a sequence of identifiers

$$S \rightarrow \epsilon \mid \text{id} \mid \text{id } S$$

- There are two parse trees for the string **id**

$$\begin{array}{l} S \rightarrow \text{id} \\ S \rightarrow \text{id } S \rightarrow \text{id} \end{array}$$

- How does this confuse the parser?

CS780(Prasad)

L16LR

40

More on Reduce/Reduce Conflicts

- Consider the states

$$\begin{array}{l} [S \rightarrow \text{id} ., \quad \$] \\ [S' \rightarrow . S, \quad \$] \quad [S \rightarrow \text{id} . S, \quad \$] \\ [S \rightarrow ., \quad \$] \Rightarrow^{\text{id}} [S \rightarrow ., \quad \$] \\ [S \rightarrow . \text{id}, \quad \$] \quad [S \rightarrow . \text{id}, \quad \$] \\ [S \rightarrow . \text{id } S, \quad \$] \quad [S \rightarrow . \text{id } S, \quad \$] \end{array}$$

- Reduce/reduce conflict** on input $\$$

$$\begin{array}{l} S' \rightarrow S \rightarrow \text{id} \\ S' \rightarrow S \rightarrow \text{id } S \rightarrow \text{id} \end{array}$$

- Better rewrite the grammar: $S \rightarrow \epsilon \mid \text{id } S$

CS780(Prasad)

L16LR

41

Strange Reduce/Reduce Conflicts

- Consider the grammar

$$\begin{array}{l} S \rightarrow P R, \quad \text{NL} \rightarrow N \mid N, \text{NL} \\ P \rightarrow T \mid \text{NL} : T \quad R \rightarrow T \mid N : T \\ N \rightarrow \text{id} \quad T \rightarrow \text{id} \end{array}$$

- P** - parameters specification
- R** - result specification
- N** - a parameter or result name
- T** - a type name
- NL** - a list of names

CS780(Prasad)

L16LR

42

Strange Reduce/Reduce Conflicts

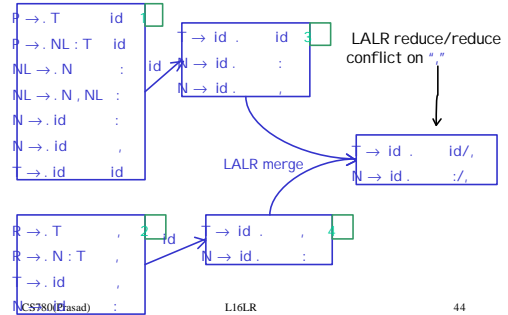
- In P an id is a
 - N when followed by $,$ or $:$
 - T when followed by id
- In R an id is a
 - N when followed by $:$
 - T when followed by $,$
- This is an LR(1) grammar.
- But it is not LALR(1). Why?
 - For obscure reasons

CS780(Prasad)

L16LR

43

A Few LR(1) States



CS780(Prasad)

L16LR

44

What Happened?

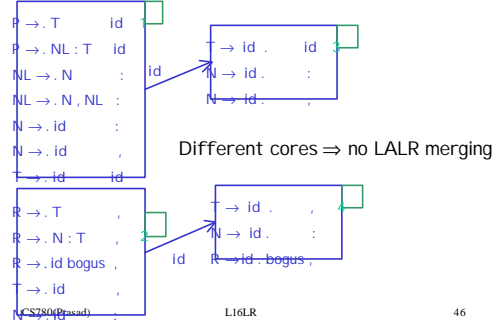
- Two distinct states were confused because they have the same core.
- Fix:** add dummy productions to distinguish the two confused states.
- E.g., add
 - $R \rightarrow id\ \text{bogus}$
 - **bogus** is a terminal not used by the lexer.
 - This production will never be used during parsing.
 - But it distinguishes R from P .

CS780(Prasad)

L16LR

45

A Few LR(1) States After Fix



CS780(Prasad)

L16LR

46

Notes on Parsing

- Parsing
 - A solid foundation: context-free grammars
 - A simple parser: LL(1)
 - A more powerful parser: LR(1)
 - An efficiency hack: LALR(1)
 - LALR(1) parser generators
- Next time we move on to semantic analysis.

CS780(Prasad)

L16LR

47