

LR Parsing

Compiler Design

CSE 504

- 1 Shift-Reduce Parsing
- 2 LR Parsers
- 3 SLR and LR(1) Parsers

Last modified: Fri Mar 06 2015 at 13:50:06 EST
Version: 1.7 16:58:46 2016/01/29
Compiled at 12:57 on 2016/02/26

Compiler Design

LR Parsing

CSE 504

1 / 32

Shift-Reduce Parsing

Leftmost and Rightmost Derivations

$$\begin{array}{lcl} E & \longrightarrow & E+T \\ E & \longrightarrow & T \\ T & \longrightarrow & \text{id} \end{array}$$

Derivations for $\text{id} + \text{id}$:

$E \Rightarrow E+T$	$E \Rightarrow E+T$
$\Rightarrow T+T$	$\Rightarrow E+\text{id}$
$\Rightarrow \text{id}+T$	$\Rightarrow T+\text{id}$
$\Rightarrow \text{id}+\text{id}$	$\Rightarrow \text{id}+\text{id}$
LEFTMOST	RIGHTMOST

Bottom-up Parsing

Given a stream of tokens w , *reduce* it to the start symbol.

$$\begin{array}{l} \hline E \longrightarrow E+T \\ E \longrightarrow T \\ T \longrightarrow \text{id} \\ \hline \end{array}$$

Parse input stream: $\text{id} + \text{id}$:

$\text{id} + \text{id}$
 $T + \text{id}$
 $E + \text{id}$
 $E + T$
 E

Reduction $\equiv$ Derivation⁻¹.

Shift-Reduce Parsing: An Example

$$\begin{array}{l} \hline E \longrightarrow E+T \\ E \longrightarrow T \\ T \longrightarrow \text{id} \\ \hline \end{array}$$

STACK	INPUT STREAM	ACTION
\$	$\text{id} + \text{id} \$$	shift
\$ id	$+ \text{id} \$$	reduce by $T \longrightarrow \text{id}$
\$ T	$+ \text{id} \$$	reduce by $E \longrightarrow T$
\$ E	$+ \text{id} \$$	shift
\$ $E +$	$\text{id} \$$	shift
\$ $E + \text{id}$	$\$$	reduce by $T \longrightarrow \text{id}$
\$ $E + T$	$\$$	reduce by $E \longrightarrow E+T$
\$ E	$\$$	ACCEPT

Handles

“A structure that furnishes a means to perform reductions”

$$\begin{array}{l} \hline E \longrightarrow E+T \\ E \longrightarrow T \\ T \longrightarrow \text{id} \\ \hline \end{array}$$

Parse input stream: `id + id`:

$$\begin{array}{c} \boxed{\text{id}} + \text{id} \\ \boxed{T} + \text{id} \\ E + \boxed{\text{id}} \\ \boxed{E + T} \\ E \end{array}$$

Handles

Handles are substrings of sentential forms:

- ① A substring that matches the right hand side of a production
- ② Reduction using that rule can lead to the start symbol
- ③ The rule forms one step in a rightmost derivation of the string

$$\begin{array}{l} E \implies \boxed{E + T} \\ \implies E + \boxed{\text{id}} \\ \implies \boxed{T} + \text{id} \\ \implies \boxed{\text{id}} + \text{id} \end{array}$$

Handle Pruning: replace handle by corresponding LHS.

Shift-Reduce Parsing

Bottom-up parsing

- **Shift:** Construct leftmost handle on top of stack
- **Reduce:** Identify handle and replace by corresponding RHS
- **Accept:** Continue until string is reduced to start symbol and input token stream is empty
- **Error:** Signal parse error if no handle is found.

Implementing Shift-Reduce Parsers

- **Stack** to hold grammar symbols (corresponding to tokens seen thus far).
- **Input stream** of yet-to-be-seen tokens.
- **Handles** appear on top of stack.
- Stack is initially empty (denoted by \$).
- Parse is successful if stack contains only the start symbol when the input stream ends.

Preparing for Shift-Reduce Parsing

- ① Identify a handle in string.
Top of stack is the *rightmost* end of the handle. What is the leftmost end?
- ② If there are multiple productions with the handle on the RHS, which one to choose?

Construct a parsing table, just as in the case of LL(1) parsing.

Shift-Reduce Parsing: Derivations

STACK	INPUT STREAM	ACTION
\$	id + id \$	shift
\$ id	+ id \$	reduce by $T \rightarrow id$
\$ T	+ id \$	reduce by $E \rightarrow T$
\$ E	+ id \$	shift
\$ E +	id \$	shift
\$ E + id	\$	reduce by $T \rightarrow id$
\$ E + T	\$	reduce by $E \rightarrow E+T$
\$ E	\$	ACCEPT

Left to Right Scan of input

Rightmost Derivation in reverse.

A Simple Example of LR Parsing

S	$\rightarrow$	BC
B	$\rightarrow$	a
C	$\rightarrow$	a

STACK	INPUT STREAM	ACTION
\$	$a\ a\ \$$	shift
\$ a	$a\ \$$	reduce by $B \rightarrow a$
\$ B	$a\ \$$	shift
\$ $B\ a$	$\$$	reduce by $C \rightarrow a$
\$ $B\ C$	$\$$	reduce by $S \rightarrow BC$
\$ S	$\$$	ACCEPT

A Simple Example of LR Parsing: A Detailed Look

S'	$\rightarrow$	S
S	$\rightarrow$	BC
B	$\rightarrow$	a
C	$\rightarrow$	a

STACK	INPUT	STATE	ACTION
\$	$a\ a\ \$$	$S' \rightarrow \bullet S$ $S \rightarrow \bullet BC$ $B \rightarrow \bullet a$	shift
\$ a	$a\ \$$	$B \rightarrow a\bullet$	reduce by 3
\$ B	$a\ \$$	$S \rightarrow B\bullet C$ $C \rightarrow \bullet a$	shift
\$ $B\ a$	$\$$	$C \rightarrow a\bullet$	reduce by 4
\$ $B\ C$	$\$$	$S \rightarrow BC\bullet$	reduce by 2
\$ S	$\$$	$S' \rightarrow S\bullet$	ACCEPT

LR Parsing: Another Example

E'	$\rightarrow$	E
E	$\rightarrow$	$E+T$
E	$\rightarrow$	T
T	$\rightarrow$	id

STACK	INPUT	STATE	ACTION
\$	$id + id \$$	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E+T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet id$	shift
$\$ id$	$+ id \$$	$T \rightarrow id \bullet$	reduce by 4
$\$ T$	$+ id \$$	$E \rightarrow T \bullet$	reduce by 3
$\$ E$	$+ id \$$	$E' \rightarrow E \bullet$ $E \rightarrow E \bullet +T$	shift
$\$ E +$	$id \$$	$E \rightarrow E+ \bullet T$ $T \rightarrow \bullet id$	shift
$\$ E + id$	$\$$	$T \rightarrow id \bullet$	reduce by 4
$\$ E + T$	$\$$	$E \rightarrow E+T \bullet$	reduce by 2
$\$ E$	$\$$	$E \rightarrow E \bullet +T$ $E' \rightarrow E \bullet$	ACCEPT

States of an LR parser

	$E' \rightarrow \bullet E$
$I_0:$	$E \rightarrow \bullet E+T$
	$E \rightarrow \bullet T$
	$T \rightarrow \bullet id$

Item: A production with “•” somewhere on the RHS. Intuitively,

- grammar symbols before the “•” are on stack;
- grammar symbols after the “•” represent symbols in the input stream.

Item set: A set of items; corresponds to a state of the parser.

States of an LR parser (contd.)

I_0	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet \text{id}$	Initial State $= \text{closure}(\{E' \rightarrow \bullet E\})$
-------	---	---

Closure:

- What other items are “equivalent” to the given item?
- Given an item $A \rightarrow \alpha \bullet B \beta$, $\text{closure}(A \rightarrow \alpha \bullet B \beta)$ is the smallest set that contains
 - 1 the item $A \rightarrow \alpha \bullet B \beta$, and
 - 2 every item in $\text{closure}(B \rightarrow \bullet \gamma)$ for every production $B \rightarrow \gamma \in G$

States of an LR parser (contd.)

I_0	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet \text{id}$	Initial State $= \text{closure}(\{E' \rightarrow \bullet E\})$
I_3	$T \rightarrow \text{id} \bullet$	$= \text{goto}(I_0, \text{id})$

Goto:

- $\text{goto}(I, X)$ specifies the next state to visit.
 - X is a terminal: when the next symbol on input stream is X .
 - X is a nonterminal: when the last reduction was to X .
- $\text{goto}(I, X)$ contains all items in $\text{closure}(A \rightarrow \alpha X \bullet \beta)$ for every item $A \rightarrow \alpha \bullet X \beta \in I$.

Collection of LR(0) Item Sets

The canonical collection of LR(0) item sets, $\mathcal{C} = \{I_0, I_1, \dots\}$ is the smallest set such that

- $\text{closure}(\{S' \rightarrow \bullet S\}) \in \mathcal{C}$.
- $I \in \mathcal{C} \Rightarrow \forall X, \text{goto}(I, X) \in \mathcal{C}$.

Canonical LR(0) Item Sets: An Example

$E' \rightarrow E$	$E \rightarrow T$
$E \rightarrow E+T$	$T \rightarrow \text{id}$

I_0	$= \text{closure}(\{E' \rightarrow \bullet E\})$	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E+T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet \text{id}$
I_1	$= \text{goto}(I_0, E)$	$E' \rightarrow E \bullet$ $E \rightarrow E \bullet +T$
I_2	$= \text{goto}(I_0, T)$	$E \rightarrow T \bullet$
I_3	$= \text{goto}(I_0, \text{id})$	$T \rightarrow \text{id} \bullet$
I_4	$= \text{goto}(I_1, +)$	$E \rightarrow E+ \bullet T$ $T \rightarrow \bullet \text{id}$
I_5	$= \text{goto}(I_4, T)$	$E \rightarrow E+T \bullet$

LR Action Table

$E' \longrightarrow E$	$E \longrightarrow T$
$E \longrightarrow E+T$	$T \longrightarrow \text{id}$

	id	+	\$
0	S, 3		
1		S, 4	A
2	R3	R3	R3
3	R4	R4	R4
4	S, 3		
5	R2	R2	R2

LR Goto Table

$E' \longrightarrow E$	$E \longrightarrow T$
$E \longrightarrow E+T$	$T \longrightarrow \text{id}$

	E	T
0	1	2
1		
2		
3		
4		5
5		

LR Parsing: States and Transitions

Action Table:

	id	+	\$
0	S, 3		
1		S, 4	A
2	R3	R3	R3
3	R4	R4	R4
4	S, 3		
5	R2	R2	R2

Goto Table:

	E	T
0	1	2
1		
2		
3		
4		5
5		

$E' \rightarrow E$	$E \rightarrow T$
$E \rightarrow E+T$	$T \rightarrow id$

STATE STACK	SYMBOL STACK	INPUT	ACTION
\$ 0	\$	id + id \$	shift, 3
\$ 0 3	\$ id	+ id \$	reduce by 4
\$ 0 2	\$ T	+ id \$	reduce by 3
\$ 0 1	\$ E	+ id \$	shift, 4
\$ 0 1 4	\$ E +	id \$	shift, 3
\$ 0 1 4 3	\$ E + id	\$	reduce by 4
\$ 0 1 4 5	\$ E + T	\$	reduce by 2
\$ 0 1	\$ E	\$	ACCEPT

LR Parser

```

while (true) {
    switch (action(state_stack.top(), current_token)) {
        case shift s':
            symbol_stack.push(current_token);
            state_stack.push(s');
            next_token();
        case reduce  $A \rightarrow \beta$ :
            pop  $|\beta|$  symbols off symbol_stack and state_stack;
            symbol_stack.push(A);
            state_stack.push(goto(state_stack.top(), A));
        case accept: return;
        default: error;
    }
}

```

LR Parsing: A review

$E' \rightarrow E$	$E \rightarrow T$
$E \rightarrow E+T$	$T \rightarrow \text{id}$

Table-driven shift reduce parsing:

Shift Move **terminal** symbols from input stream to stack.

Reduce Replace top elements of stack that form an instance of the RHS of a production with the corresponding LHS

Accept Stack top is the start symbol when the input stream is exhausted

Table constructed using LR(0) Item Sets.

SLR and LR(1) Parsers

Conflicts in Parsing Table

Grammar:

$S' \rightarrow S$
$S \rightarrow a S$
$S \rightarrow \epsilon$

Item Sets:

I_0	$= \text{closure}(\{S' \rightarrow \bullet S\})$	$S' \rightarrow \bullet S$ $S \rightarrow \bullet a S$ $S \rightarrow \bullet$
I_1	$= \text{goto}(I_0, S)$	$S' \rightarrow S \bullet$
I_2	$= \text{goto}(I_0, a)$	$S \rightarrow a \bullet S$ $S \rightarrow \bullet a S$ $S \rightarrow \bullet$
I_3	$= \text{goto}(I_2, S)$	$S \rightarrow a S \bullet$

Action Table:

	a	\$
0	$S, 2$ $R 3$	R 3
1		A
2	$S, 2$ $R 3$	R 3
3	R 2	R 2

Shift-Reduce Conflict

“Simple LR” (SLR) Parsing

Constructing Action Table *action*, indexed by *states* $\times$ *terminals*, and Goto Table *goto*, indexed by *states* $\times$ *nonterminals*:

- Construct $\{I_0, I_1, \dots, I_n\}$, the LR(0) sets of items for the grammar.
For each i , $0 \leq i \leq n$, do the following:
- If $A \rightarrow \alpha \bullet a \beta \in I_i$, and $goto(I_i, a) = I_j$, set $action[i, a] = \text{shift } j$.
- If $A \rightarrow \gamma \bullet \in I_i$ (A is not the start symbol),
for each $a \in FOLLOW(A)$, set $action[i, a] = \text{reduce } A \rightarrow \gamma$.
- If $S' \rightarrow S \bullet \in I_i$, set $action[i, \$] = \text{accept}$.
- If $goto(I_i, A) = I_j$ (A is a nonterminal), set $goto[i, A] = j$.

SLR Parsing Table

Grammar:

S'	$\rightarrow$	S
S	$\rightarrow$	$a S$
S	$\rightarrow$	ϵ

$$FOLLOW(S) = \{\$, \epsilon\}$$

Item Sets:

I_0	$= \text{closure}(\{S' \rightarrow \bullet S\})$	$S' \rightarrow \bullet S$ $S \rightarrow \bullet a S$ $S \rightarrow \bullet$
I_1	$= \text{goto}(I_0, S)$	$S' \rightarrow S \bullet$
I_2	$= \text{goto}(I_0, a)$	$S \rightarrow a \bullet S$ $S \rightarrow \bullet a S$ $S \rightarrow \bullet$
I_3	$= \text{goto}(I_2, S)$	$S \rightarrow a S \bullet$

SLR Action Table:

	a	$\$$
0	S, 2	R 3
1		A
2	S, 2	R 3
3		R 2

Deficiencies of SLR Parsing

- SLR(1) treats all occurrences of a RHS on stack as identical.
- Only a few of these reductions may lead to a successful parse.
- Example:

$$\begin{array}{c} \hline S \longrightarrow AaAb \quad A \longrightarrow \epsilon \\ S \longrightarrow BbBa \quad B \longrightarrow \epsilon \\ \hline \end{array}$$

$$I_0 = \{[S' \rightarrow \bullet S], [S \rightarrow \bullet AaAb], [S \rightarrow \bullet BbBa], [A \rightarrow \bullet], [B \rightarrow \bullet]\}.$$

- Since $FOLLOW(A) = FOLLOW(B)$, we have reduce/reduce conflict in state 0.

LR(1) Item Sets

Construct LR(1) items of the form $A \longrightarrow \alpha \bullet \beta, a$, which means:

The production $A \longrightarrow \alpha\beta$ can be applied when the next token on input stream is a .

$$\begin{array}{c} \hline S \longrightarrow AaAb \quad A \longrightarrow \epsilon \\ S \longrightarrow BbBa \quad B \longrightarrow \epsilon \\ \hline \end{array}$$

An example LR(1) item set:

$$I_0 = \{[S' \rightarrow \bullet S, \$], [S \rightarrow \bullet AaAb, \$], [S \rightarrow \bullet BbBa, \$], [A \rightarrow \bullet, a], [B \rightarrow \bullet, b]\}.$$

LR(1) and LALR(1) Parsing

LR(1) parsing: Parse tables built using LR(1) item sets.

LALR(1) parsing: *Look Ahead* LR(1)

- Merge LR(1) item sets; then build parsing table.
- Typically, LALR(1) parsing tables are much smaller than LR(1) parsing table.
- $SLR(1) \subset LALR(1) \subset LR(1)$.
- $LL(1) \not\subset SLR(1)$, but $LL(1) \subset LR(1)$.

YACC

Yet Another Compiler Compiler:
LALR(1) parser generator.

- Grammar rules written in a specification (.y) file, analogous to the regular definitions in a lex specification file.
- Yacc translates the specifications into a parsing function `yyparse()`.

$$\text{spec.y} \xrightarrow{\text{yacc}} \text{spec.tab.c}$$

- `yyparse()` calls `yylex()` whenever input tokens need to be consumed.
- `bison`: GNU variant of `yacc`.
- `ply`: Python's "yacc"; provides function `yacc()` that is similar to Yacc's `yyparse()`

Using Yacc

```
%{
    ... C headers (#include)
}%

... Yacc declarations:
    %token ...
    %union{...}
    precedences
%%
... Grammar rules with actions:

Expr:  Expr TOK_PLUS Expr
      | Expr TOK_STAR Expr
      ;
%%
... C support functions
```

Parsing in PLY

See <http://www.dabeaz.com/ply/ply.html>

```
import ply.yacc as yacc
... import tokens from PLY/lexer

# precedences:
precedence = ( ('left', 'TOK_PLUS'), ('left', 'TOK_STAR') )

# Grammar rules with actions:
def p_expression_plus(p):
    'expr: expr TOK_PLUS expr'
    pass #action, if necessary

def p_expression_minus(p):
    'expr: expr TOK_STAR expr'
    pass #action, if necessary
```