LR Parsing

Petr Horáček, Eva Zámečníková, Ivana Burgetová and Alexander Meduna

Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 00 Brno, CZ

FRVŠ MŠMT FR0097/2011/G1

Outline

- **LR Parsing Algorithm**
- **Construction of LR Table**
- **Handling Errors in LR Parsing**

LR Parsing

LR Parsers

- ***L**eft-to-right* scan of tokens
- ***R**ightmost* derivation
- Uses right parse – reverse sequence of rules
- Bottom-up parsing
- Based on *LR tables* constructed from *LR grammars*
    - LR grammar – context-free grammar for which LR table can be built

Advantages

- LR parsers are fast
- Easy way of handling syntax errors
- Ultimately powerful
    - The family of LR languages equals the family of languages accepted by deterministic pushdown automata (DPDA)

Topic

- **LR Parsing Algorithm**

- **Construction of LR Table**

- **Handling Errors in LR Parsing**

LR Parsing Algorithm

LR table

Consider LR grammar $G = (N, T, P, S)$. Then G-based **LR table** consists of:

- G-based action part ${}_G action$
- G-based goto part ${}_G goto$

- Rows are denoted by the symbols of ${}_G\Theta = \{\theta_1, \dots, \theta_m\}$
  - States of extended pushdown automata (LR parser is EPDA)
- Columns of ${}_G action$ are denoted by the symbols of T
  - Terminal symbols
- Columns of ${}_G goto$ are denoted by the symbols of N
  - Nonterminal symbols

Configuration of the parser

$$\triangleright q_0 Y_1 q_1 \dots Y_{m-1} q_{m-1} Y_m q_m \diamond v \triangleleft$$

where $q_i \in {}_G\Theta$, $Y_i \in N \cup T$, $v \in suffixes(w)$, $w \in L(G)$

goto and *action* Part of LR Table

Table: $_{G}action$

	t_1	$\dots$	t_i	$\dots$	t_n
θ_1	$action[\theta_j, t_i] \in {_G}\Theta \cup P \cup \{ ☺ \}$ or blank				
$\vdots$					
θ_1					
$\vdots$					
θ_m					

Table: $_{G}goto$

	A_1	$\dots$	A_i	$\dots$	A_k
θ_i	$goto[\theta_j, A_i] \in N$ or blank				
$\vdots$					
θ_j					
$\vdots$					
θ_m					

Operations LR-REDUCE and LR-SHIFT

LR-REDUCE

If

- $p\colon A \to X_1 X_2 \dots X_n \in P$
  - for some $n \geq 0$, $X_j \in N \cup T$, $1 \leq j \leq n$
- $o_0 X_1 o_1 X_2 o_2 \dots o_{n-1} X_n o_n$ is the pushdown top
  - o_n topmost, $o_k \in {}_G\Theta, 0 \leq k \leq n$

then **LR-REDUCE**(p) replaces $o_0 X_1 o_1 X_2 o_2 \dots o_{n-1} X_n o_n$ with Ah on the pushdown top

- $h \in {}_G\Theta$ is defined as $h = {}_G goto[o_0, A]$, otherwise **REJECT**

LR-SHIFT

- Let $ins_1 = t, t \in N \cup T$ and $action[pd_1, t] = o, o \in {}_G\Theta$
- **LR-SHIFT** extends pushdown *pd* by *to* and advances to the next input
  - *to* now occurs at the top of the pushdown (*o* is the topmost) and ins_1 refers to the input symbol occurring right behind t in the input string

Algorithm 1.1: LR Parser

- **Input**: An LR grammar, $G = (N, T, P, S)$,
  an input string w, $w \in T^*$ and
  G-based LR table.
- **Output: ACCEPT** if $w \in L(G)$, or
  REJECT if $w \notin L(G)$.

Method

```
$pd := \triangleright\theta_1$
repeat
   case action[$pd_1$, $ins_1$] of
      in ${}_G\Theta$: LR-SHIFT
      in $P$: LR-REDUCE($p$) with $p = action[pd_1, ins_1]$
      $\Box$ : REJECT {$\Box$ denotes blank symbol (undefined action)}
      ☺ : ACCEPT
   end case
until ACCEPT or REJECT
```

LR Table Example

- Consider grammar G with the following rules:

$$1: S \to S \vee A \quad 2: S \to A \quad 3: A \to A \wedge B$$
$$4: A \to B \quad 5: B \to (S) \quad 6: B \to i$$

where S is the start symbol, $T = \{\vee, \wedge, (,), i\}$ and $N = \{A, B\}$

Table: G-based LR table example

	$\wedge$	$\vee$	i	(	)	$\triangleleft$	S	A	B
θ_1			θ_6	θ_5			θ_2	θ_3	θ_4
θ_2		θ_7				☺			
θ_3	θ_8	2			2	2			
θ_4	4	4			4	4			
θ_5			θ_6	θ_5			θ_9	θ_3	θ_4
θ_6	6	6			6	6			
θ_7			θ_6	θ_5				θ_{10}	θ_4
θ_8			θ_6	θ_5					θ_{11}
θ_9		θ_7			θ_{12}				
θ_{10}	θ_7	1			1	1			
θ_{11}	3	3			3	3			
θ_{12}	5	5			5	5			
	action part						goto part		

LR Table Example

- Consider an expression

$$i \wedge i \in L(G)$$

- We make a parse by Algorithm 1.1
- The sequence of configurations is given in following table

Configuration	Table Entry	Parsing Action
$\rhd\theta_1\diamond i \wedge i\lhd$	action$[\theta_1, i]$=θ_6	**LR-SHIFT**(i)
$\rhd\theta_1 i\theta_6\diamond \wedge i\lhd$	action$[\theta_6, \wedge]$=6, goto $[\theta_1, B] = \theta_4$	**LR-REDUCE**(6)
$\rhd\theta_1 B\theta_4\diamond \wedge i\lhd$	$action[\theta_4, \wedge] = 4, goto[\theta_1, A] = \theta_3$	**LR-REDUCE**(4)
$\rhd\theta_1 A\theta_3\diamond \wedge i\lhd$	$action[\theta_3, \wedge] = \theta_8$	**LR-SHIFT**$(\vee)$
$\rhd\theta_1 A\theta_3 \wedge \theta_8\diamond i\lhd$	$action[\theta_8, i] = \theta_8$	**LR-SHIFT**(i)
$\rhd\theta_1 A\theta_3 \wedge \theta_8 i\theta_6\diamond\lhd$	$action[\theta_6, \lhd] = 6, goto[\theta_8, B] = \theta_{11}$	**LR-REDUCE**(6)
$\rhd\theta_1 A\theta_3 \wedge \theta_8 B\theta_{11}\diamond\lhd$	$action[\theta_{11}, \lhd] = 3, goto[\theta_1, A] = \theta_3$	**LR-REDUCE**(3)
$\rhd\theta_1 A\theta_3\diamond\lhd$	$action[\theta_3, \lhd] = 2, goto[\theta_1, S] = \theta_2$	**LR-REDUCE**(2)
$\rhd\theta_1 S\theta_2\diamond\lhd$	**ACCEPT**	**ACCEPT**

Topic

- LR Parsing Algorithm
- **Construction of LR Table**
- Handling Errors in LR Parsing

Construction of ${}_G\Theta$ – Items

Item

$$A \rightarrow x\diamond y$$

for each rule $A \rightarrow z$ and any two strings x and y such that $z = xy$

- x – handle prefix on the *pd* top
- Start item: $A \rightarrow \diamond z$
- End item: $A \rightarrow z\diamond$

Example

- Rule: $S \rightarrow S \vee A$
- Items: $S \rightarrow \diamond S \vee A, S \rightarrow S\diamond \vee A, S \rightarrow S \vee \diamond A, S \rightarrow S \vee A\diamond$

Convention

- ${}_GI$ – set of all items for LR grammar G
- ${}_GI_{start}$ – set of start items, ${}_GI_{start} \subseteq {}_GI$
- ${}_GI_{end}$ – set of end items, ${}_GI_{end} \subseteq {}_GI$
- ${}_G\Omega = 2^{{}_GI}$ – state space

Construction of ${}_{G}\Theta$ – Idea I

1. Change the start symbol S to a new start symbol Z in G, and add a dummy rule $Z \rightarrow S$
    - Every derivation in G now starts by applying $Z \rightarrow S$
2. Initially, set ${}_{G}\Theta = \emptyset$, ${}_{G}W = \{\{Z \rightarrow \diamond S\}\}$
    - ${}_{G}W$ – auxiliary item set
3. Repeat extensions I and II until no new item set can be included in ${}_{G}W$

Construction of ${}_G\Theta$ – Idea II

Extension I

- Let $I \in {}_GW$. Suppose that u appears on the *pd* top, and let $A \to uBv \in P$
- Observe: if $A \to u\diamond Bv \in I$ and $B \to \diamond z \in {}_GI_{start}$, then by using $B \to z$, the parser can reduce z to B
  - Does not affect u on the *pd* top because $B \to \diamond z$ is a start item
- Thus, add $B \to \diamond z$ into I
- Repeat until I can no longer be extended in this way
- Add the resulting I to ${}_G\Theta$

```
repeat
   if $A \to u\diamond Bv \in I$ and $B \to z \in {}_GR$ then
      include $B \to \diamond z$ into $I$
   end if
until no change
include $I$ into ${}_G\Theta$
```

Construction of ${}_G\Theta$ – Idea III

Extension II

- Based upon a relation ${}_G\circlearrowleft$ from ${}_G\Omega \times (N \cup T)$ to ${}_G\Omega$:

$$ {}_G\circlearrowleft(I, X) = \{A \to uX\diamond v | A \to u\diamond Xv \in I, A \in N, u, v \in N \cup T\} $$

- Let $I \in {}_GW$ and $A \to uX\diamond v \in I$
- Consider a part of rightmost derivation in G in reverse order, during which a portion of the input string is reduced to X – simulating this part, the parser obtains X on the pushdown
- Thus, for every $I \in {}_GW$ and $X \in N \cup T$, extend ${}_GW$ by ${}_G\circlearrowleft(I, X)$ unless ${}_G\circlearrowleft(I, X)$ is empty

```
for all $X \in N \cup T$ with ${}_G\circlearrowleft(I, X) \neq \emptyset$ do
    include ${}_G\circlearrowleft(I, X)$ into ${}_GW$
end for
```

Algorithm 2.1: Construction of ${}_{G}\Theta$

- **Input:** An LR grammar, $G = (N, T, P, S)$, extended by the dummy rule $Z \rightarrow S$, where Z is the new start symbol.
- **Output:** ${}_{G}\Theta$.
- **Note:** An auxiliary set ${}_{G}W \subseteq {}_{G}\Omega$ is used.

Method

```
set $_{G}W = \{\{Z \rightarrow \diamond S\}\}$
set $_{G}\Theta = \emptyset$
repeat
    for all $I \in {}_{G}W$ do
        repeat {start of extension I}
            if $A \rightarrow u\diamond Bv \in I$ and $B \rightarrow z \in P$ then
                include $B \rightarrow \diamond z$ into $I$
            end if
        until no change
        include $I$ into $_{G}\Theta$
        for all $X \in N \cup T$ with $_{G}\circlearrowleft(I, X) \neq \emptyset$ do {start of extension II}
            include $_{G}\circlearrowleft(I, X)$ into $_{G}W$
        end for
    end for
until no change
```

Construction of ${}_G\Theta$ – Example

Example

- Consider ${}_{cond}G$. Add a dummy rule $Z \rightarrow S$ and define Z as the start symbol

$$\begin{array}{llll} 0: Z \rightarrow S & 1: S \rightarrow S \vee A & 2: S \rightarrow A & 3: A \rightarrow A \wedge B \\ 4: A \rightarrow B & 5: B \rightarrow (S) & 6: B \rightarrow i & \end{array}$$

- Apply Algorithm 2.1. First, set ${}_{cond}{}_G\Theta = \emptyset$, ${}_GW = \{\{Z \rightarrow \diamond S\}\}$
- By extension I, extend $\{Z \rightarrow \diamond S\} \in {}_GW$ to: $\{Z \rightarrow \diamond S, S \rightarrow \diamond S \vee A, S \rightarrow \diamond A, A \rightarrow \diamond A \wedge B, A \rightarrow \diamond B, B \rightarrow \diamond(S), B \rightarrow \diamond i\}$
- For $I = \{Z \rightarrow \diamond S, S \rightarrow S \vee A\}$, we have ${}_G\circlearrowleft(I, S) = \{Z \rightarrow S\diamond, S \rightarrow S\diamond \vee A\}$
- Thus, by extension II, include $\{Z \rightarrow S\diamond, S \rightarrow S\diamond \vee A\}$ into ${}_GW$
- Perform second iteration of I and II, and so on

Construction of ${}_G\ominus$ – Example

Rules

$0\colon Z \rightarrow S$ $1\colon S \rightarrow S \vee A$ $2\colon S \rightarrow A$ $3\colon A \rightarrow A \wedge B$
$4\colon A \rightarrow B$ $5\colon B \rightarrow (S)$ $6\colon B \rightarrow i$

${}_{cond}G^{\ominus}$	**Item Sets**
θ_1	$\{Z \rightarrow \diamond S, S \rightarrow \diamond S \vee A, S \rightarrow \diamond A, A \rightarrow \diamond A \wedge B, A \rightarrow \diamond B, B \rightarrow \diamond (S), B \rightarrow \diamond i\}$
θ_2	$\{Z \rightarrow S\diamond, S \rightarrow S\diamond \vee A\}$
θ_3	$\{S \rightarrow A\diamond, A \rightarrow A\diamond \wedge B\}$
θ_4	$\{A \rightarrow B\diamond\}$
θ_5	$\{B \rightarrow (\diamond S), S \rightarrow \diamond S \vee A, S \rightarrow \diamond A, A \rightarrow \diamond A \wedge B, A \rightarrow \diamond B, B \rightarrow \diamond (S), B \rightarrow \diamond i\}$
θ_6	$\{B \rightarrow i\diamond\}$
θ_7	$\{S \rightarrow S \vee \diamond A, A \rightarrow \diamond A \wedge B, A \rightarrow \diamond B, B \rightarrow \diamond (S), B \rightarrow \diamond i\}$
θ_8	$\{A \rightarrow A \wedge \diamond B, B \rightarrow \diamond (S), B \rightarrow \diamond i\}$
θ_9	$\{B \rightarrow (S\diamond), S \rightarrow S\diamond \vee A\}$
θ_{10}	$\{S \rightarrow S \vee A\diamond, A \rightarrow A\diamond \wedge B\}$
θ_{11}	$\{A \rightarrow A \wedge B\diamond\}$
θ_{12}	$\{B \rightarrow (S)\diamond\}$

Construction of LR Table

I. *goto* part

- Consider item $A \rightarrow u\diamond Bv$, where $I \in {}_G\Theta$, $A, B \in N$ and $u, v \in N \cup T$
- After reducing portion of the input string to B, parser extends the prefix u by B, so uB occurs on the *pd* top

```
if ${}_G\circlearrowleft(\theta_i, B) = \theta_j - {}_G I_{start}$, where $B \in N$ then
    $goto[\theta_i, B] = \theta_j$
end if
```

II. *action* part – shift

- By analogy with I

```
if ${}_G\circlearrowleft(\theta_i, b) = \theta_j - {}_G I_{start}$, where $b \in T$ then
    $action[\theta_i, b] = \theta_j$
end if
```

Construction of LR Table

III. *action* part – reduction

- Consider a rule $p\colon A \to u \in P$ and $A \to u\diamond \in {}_G I_{end}$
    - A complete handle u on *pd* top
- Parser reduces u to A provided that after the reduction, A is followed by terminal a that may legally occur after A in a sentential form

```
**if** $A \to u\diamond \in \theta_i, a \in follow(A), p\colon A \to u \in P$ **then**
    $action[\theta_i, a] = p$
**end if**
```

- Note that:
    - Every derivation starts with $0\colon Z \to S$
    - LR parser simulates rightmost derivations in reverse
    - Input symbol $\lhd$ – all the input has been read
- Thus, if $Z \to S\diamond \in \theta_i$, set $action[\theta_i, \lhd] = ☺$ (parsing completed successfully)

```
**if** $Z \to S\diamond \in \theta_i$ **then**
    $action[\theta_i, \lhd] = ☺$
**end if**
```

Algorithm 2.2: LR Table (1/2)

- **Input:** An LR grammar $G = (N, T, P, S)$, in which Z and $0 : Z \to S$ have the same meaning as in Algoritm 2.1, and ${}_G\Theta$ constructed by Algorithm 2.1.
- **Output:** A G-based LR table, consisting of the *action* and *goto* parts.
- **Note:** We suppose that $A, B \in N$, $b \in T$ and $u, v \in (N \cup T)^*$ in this algorithm.

Method

denote the rows of *action* and *goto* with the members of ${}_G\Theta$
denote the columns of *action* and *goto* with the members of T and N, respectively

{continued on next slide}

Algorithm 2.2: LR Table (2/2)

Method (cont.)

```
repeat
  for all $\theta_i, \theta_j \in {}_G\Theta$ do
    if ${}_G\circlearrowleft(\theta_i, B) = \theta_j - {}_GI_{start}$, where $B \in N$ then
      $goto[\theta_i, B] = \theta_j$
    end if
    if ${}_G\circlearrowleft(\theta_i, b) = \theta_j - {}_GI_{start}$, where $b \in T$ then
      $action[\theta_i, b] = \theta_j$
    end if
    if $A \to u\diamond \in \theta_i \cap {}_GI_{end}, a \in follow(A), i : A \to u \in P$ then
      $action[\theta_i, a] = i$
    end if
  end for
until no change
if $Z \to S\diamond \in \theta_i$ then
  $action[\theta_i, \lhd] =$ ☺ {success}
  {all the other entries remain blank and, thereby, signalize
  a syntax error}
end if
```

Construction of LR Table – Example

Example

- Consider again ${}_{cond}G$

$$
\begin{array}{llll}
0: Z \to S & 1: S \to S \vee A & 2: S \to A & 3: A \to A \wedge B \\
4: A \to B & 5: B \to (S) & 6: B \to i &
\end{array}
$$

- Consider ${}_{cond}G\Theta = \{\theta_1, \theta_2, \ldots, \theta_{12}\}$ (obtained in previous example)
- According to the first **if** statement in Algorithm 2.2, $goto[\theta_1, S] = \theta_2$ because $S \to \diamond S \vee A \in \theta_1$ and $S \to S\diamond \vee A \in \theta_2$
- Second **if** statement: $action[\theta_2, \vee] = \theta_7$ because $S \to S\diamond \vee A \in \theta_2$ and $S \to S \vee \diamond A \in \theta_7$
- Third **if** statement: $action[\theta_{10}, \vee] = 2$ because $2 : S \to A\diamond \in \theta_{10}$ and $\vee \in follow(A)$
- Repeat until there is no change
- Set $action[\theta_2, \lhd] = ☺$ because θ_2 contains $Z \to S\diamond$

Construction of LR Table – Example

Table: *G*-based LR table example

	$\wedge$	$\vee$	i	(	)	$\lhd$	S	A	B
θ_1			θ_6	θ_5			θ_2	θ_3	θ_4
θ_2		θ_7				☺			
θ_3	θ_8	2			2	2			
θ_4	4	4			4	4			
θ_5			θ_6	θ_5			θ_9	θ_3	θ_4
θ_6	6	6			6	6			
θ_7			θ_6	θ_5				θ_{10}	θ_4
θ_8			θ_6	θ_5					θ_{11}
θ_9		θ_7			θ_{12}				
θ_{10}	θ_7	1			1	1			
θ_{11}	3	3			3	3			
θ_{12}	5	5			5	5			
	action part						goto part		

Topic

- LR Parsing Algorithm
- Construction of LR Table
- **Handling Errors in LR Parsing**

LR Parsing: Handling Errors

Error detection

No valid continuation for the portion of the input thus far scanned

- More exact than in precedence parsing
- Detection of all possible errors by using *action part*
  - We can reduce the size of *goto part* by removing unneeded blank entries

LR error recovery methods

- Panic-mode LR Error Recovery
- Ad-hoc Recovery

Panic-mode LR Error Recovery

Method

- Try to isolate **the shortest** possible erroneous substring,
- skip this substring, and
- resume parsing process

- Basic idea of this method: we have selected set of nonterminals ${}_{G}O$ representing major pieces of program such as expressions or statements
- Find the shortest string uv, where:
  - $u \in (N \cup T)^*$ is obtained from the current pushdown top $x \in ((N \cup T)_{G}\Theta)^*$ by deletion of all pushdown symbols
  - v is the shortest input prefix followed by input symbol a from $follow(A)$, where$A \in O$ and $A_{rm} \Rightarrow^* uv$
- Let x be preceded by $o \in {}_{G}\Theta$ and $goto[o, A] = \theta$
- To recover, this method replaces x with $A\theta$ on the pushdown and skips the input prefix v
- After this it resumes the parsing process from $action[\theta, a]$

Ad-hoc Recovery

- Resembles the way the precedence parser handles the table-detected errors
- This method considers **each** blank *action* entry, which signalize error
- We decide the most probable mistake that led to particular error and according to this we design recovery procedure
- Typical recovery routines: modify the pushdown or input by *changing, inserting* or *deleting* some symbols
- Modification **has to** avoid infinite loops
- Each blank entry is filled with the reference to the corresponding recovery routine

Ad-hoc Recovery Example

- Consider again the grammar G:

$$\begin{array}{lll} 1: S \rightarrow S \vee A & 2: S \rightarrow A & 3: A \rightarrow A \wedge B \\ 4: A \rightarrow B & 5: B \rightarrow (S) & 6: B \rightarrow i \end{array}$$

where S is the start symbol, $T = \{\vee, \wedge, (,), i\}$ and $N = \{A, B\}$

- As an expression we take

$$i \vee)$$

- The parsing process for this input is interrupted after six steps ⇒ **RECOVERY**
- We update the *action* part of table by filling the blank entries by recovery routines, the *goto* part of LR table stays the same
- The construction of recovery procedures needs sophisticated approach

Ad-hoc Recovery Example

Table: *G*-based LR table example

	$\wedge$	$\vee$	i	(	)	$\lhd$
θ_1	①	①	θ_6	θ_5	②	①
θ_2	①	θ_7	③	③	②	☺
θ_3	θ_8	2	③	③	2	2
θ_4	4	4	③	③	4	4
θ_5	①	①	θ_6	θ_5	②	①
θ_6	6	6	③	③	6	6
θ_7	①	①	θ_6	θ_5	②	①
θ_8	①	①	θ_6	θ_5	②	①
θ_9	①	θ_7	③	③	θ_{12}	①
θ_{10}	θ_7	1	③	③	1	1
θ_{11}	3	3	③	③	3	3
θ_{12}	5	5	③	③	5	5

Ad-hoc Recovery Example

- The description of recovery procedures ① through ④
- Consider string $i \vee ($ as an input

① **diagnostic**: missing i or (, **recovery**: insert $i\theta_6$ onto the pushdown
② **diagnostic**: unbalanced), **recovery**: delete the input)
③ **diagnostic**: missing operator, **recovery**: insert $\vee\theta_5$ onto the pushdown
④ **diagnostic**: missing), **recovery**: insert $)\theta_6$ onto the pushdown

Then we can make LR parse. After the input is finaly accepted there are saved error reports with the information about used recovery processes.