Syntax Analysis: Simple LR Parsing

Lecture 6

Objectives

By the end of this lecture you should be able to:

1. Identify LR(0) items.
2. Construct an LR(0) automaton for a CFG.
3. Construct the SLR parsing table for a CFG.
4. Trace the operation of an SLR parser.

Outline

Outline

What is LR(k) Parsing?

Definition

An LR grammar is a grammar for which a deterministic shift-reduce parser may be constructed.

- An LR(k) parser is such a deterministic shift-reduce parser.
- LR(k) stands for left-to-right input scanning in a right-most derivation with k input symbols of lookahead.
- We shall be interested in cases where $k \leq 1$.

What is LR(k) Parsing?

Definition

An LR grammar is a grammar for which a deterministic shift-reduce parser may be constructed.

- An LR(k) parser is such a deterministic shift-reduce parser.
- LR(k) stands for left-to-right input scanning in a right-most derivation with k input symbols of lookahead.
- We shall be interested in cases where $k \leq 1$.

Why LR Parsers?

1. LR parsers can be constructed to recognize almost all context-free constructs in programming languages.
2. Efficient implementations of LR parsers are possible.
3. The set of LR grammars is a proper superset of the set of LL grammars.

Grammar G_6

Example

We shall often refer to the following grammar G_6.

$$\begin{array}{lcl} E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & T * F \mid F \\ F & \longrightarrow & (\ E\ ) \mid \mathbf{id} \end{array}$$

Problems with Shift-Reduce Parsing (I)

One problem with shift-reduce parsers we have seen so far is that it is always possible to shift if there are symbols available in the input.

Example

- With G_6 and input **id**$*$**id**, having shifted **id**, a shift-reduce parser may decide to shift $*$.
- But clearly, given the rules of G_6, this will never succeed.

Problems with Shift-Reduce Parsing (I)

One problem with shift-reduce parsers we have seen so far is that it is always possible to shift if there are symbols available in the input.

Example

- With G_6 and input **id**$*$**id**, having shifted **id**, a shift-reduce parser may decide to shift $*$.
- But clearly, given the rules of G_6, this will never succeed.

Problems with Shift-Reduce Parsing (II)

In shift-reduce parsing we can always reduce if the right side of a production appears on top of the stack.

Example

- With G_6 and input **id**$*$**id**, we may reach a configuration where T appears on top of the stack and $*$**id** remains in the input stream.
- We can choose to reduce using the rule $E \rightarrow T$.
- But clearly, given the rules of G_6, this will never succeed.

Can we avoid wrong decisions, especially that we know better?

Problems with Shift-Reduce Parsing (II)

In shift-reduce parsing we can always reduce if the right side of a production appears on top of the stack.

Example

- With G_6 and input **id**$*$**id**, we may reach a configuration where T appears on top of the stack and $*$**id** remains in the input stream.
- We can choose to reduce using the rule $E \rightarrow T$.
- But clearly, given the rules of G_6, this will never succeed.

Can we avoid wrong decisions, especially that we know better?

Problems with Shift-Reduce Parsing (II)

In shift-reduce parsing we can always reduce if the right side of a production appears on top of the stack.

Example

- With G_6 and input **id**$*$**id**, we may reach a configuration where T appears on top of the stack and $*$**id** remains in the input stream.
- We can choose to reduce using the rule $E \rightarrow T$.
- But clearly, given the rules of G_6, this will never succeed.

Can we avoid wrong decisions, especially that we know better?

Outline

LR(0) Items

Definition

An LR(0) item of CFG $G = \langle V, \Sigma, R, S \rangle$ is a pair $\langle A \rightarrow \alpha, i \rangle$, where $(A \rightarrow \alpha) \in R$ and $0 \leq i \leq |\alpha|$.

- Intuitively, an LR(0) item is a rule and a position in the right side of the rule.
- Rather than using the ordered-pair notation, we represent items by a rule, with a dot (".") added somewhere to its right side.
  - Thus, $\langle A \rightarrow \mathtt{a}B\mathtt{b}, 2 \rangle \equiv A \rightarrow \mathtt{a}B.\mathtt{b}$

The LR(0) NFA

Definition

For a CFG $G = \langle V, \Sigma, R, S \rangle$, the LR(0) NFA is an NFA $N_G = \langle I, V \cup \Sigma, \delta, S' \rightarrow .S, I \rangle$, where

- I is the set LR(0) items of G together with $S' \rightarrow .S$;
- $S' \notin V \cup \Sigma$;
- $\delta(A \rightarrow \alpha.s\beta, s) = \{A \rightarrow \alpha s.\beta\}$;
- $\delta(A \rightarrow \alpha.B\beta, \varepsilon) = \{B \rightarrow .\gamma \mid (B \rightarrow \gamma) \in R\}$

The LR(0) Automaton

Definition

The LR(0) automaton for a CFG G is the DFA M_G which is equivalent to N_G and constructed using the standard subset construction.

- Note that constructing M_G amounts to computing the ε-closures of states of N_G.
- The language of M_G (and N_G) is the set of all sentential forms that are allowed to appear on top of the stack of a shift-reduce parser.
  - Thus, if other sentential forms appear on top of the stack, parsing fails.

Exercise

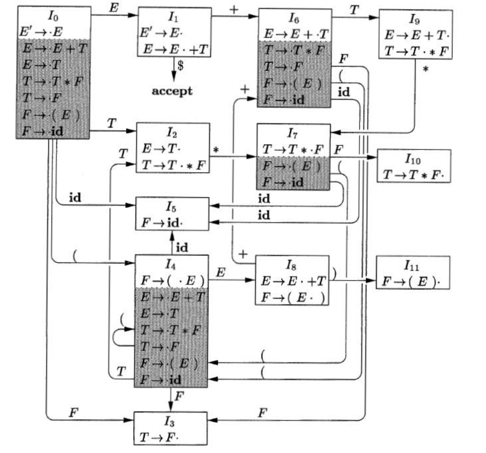
Example

Construct the LR(0) automaton of G_6:

$$
\begin{array}{lcl}
E & \longrightarrow & E + T \mid T \\
T & \longrightarrow & T * F \mid F \\
F & \longrightarrow & (\ E\ ) \mid \textbf{id}
\end{array}
$$

Exercise (II)

Example



© Aho et al. (2007)

Outline

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or $ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or $ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or \$ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or \$ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or \$ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or \$ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

The LR Parsing Table

- LR parsers all use a parsing table to guide their decisions.
- The parsing table is really two tables:
  1. The Action Table: Associates with each LR(0) automaton state and terminal symbol or \$ an action to be performed.
     - Actions are one of: shift, reduce, accept, or error.
  2. The Goto Table: Associates with each LR(0) automaton state and nonterminal symbol an LR(0) automaton state.
- The method used to construct the table yields different types of LR parsers.
- We first consider simple LR parsers (SLR parsers).

Constructing the SLR Parsing Table

We are given a CFG $G = \langle V, \Sigma, R, S \rangle$.

1. Construct M_G.
2. For all states q of M_G
    1. $\text{GOTO}(q, A) = \delta(q, A)$, for every $A \in V$.
    2. If $a \in \Sigma$ and $(A \longrightarrow \alpha.a\beta) \in q$, then $\text{ACTION}(q, a) =$ "shift $\delta(q, a)$".
    3. If $A \neq S'$, $a \in Follow(A)$, and $(A \longrightarrow \alpha.) \in q$, then $\text{ACTION}(q, a) =$ "reduce $A \longrightarrow \alpha$".
    4. If $(S' \longrightarrow S.) \in q$, then $\text{ACTION}(q, \$) =$ "accept".
    5. Otherwise $\text{ACTION}(q, a) =$ "error".

If any conflicting actions result from the above construction, we say that G is not SLR.

Constructing the SLR Parsing Table

We are given a CFG $G = \langle V, \Sigma, R, S \rangle$.

1. Construct M_G.
2. For all states q of M_G
   1. $\text{GOTO}(q, A) = \delta(q, A)$, for every $A \in V$.
   2. If $a \in \Sigma$ and $(A \rightarrow \alpha.a\beta) \in q$, then $\text{ACTION}(q, a) =$ "shift $\delta(q, a)$".
   3. If $A \neq S'$, $a \in \mathit{Follow}(A)$, and $(A \rightarrow \alpha.) \in q$, then $\text{ACTION}(q, a) =$ "reduce $A \rightarrow \alpha$".
   4. If $(S' \rightarrow S.) \in q$, then $\text{ACTION}(q, \$)$ = "accept".
   5. Otherwise $\text{ACTION}(q, a)$ = "error".

If any conflicting actions result from the above construction, we say that G is not SLR.

Exercise (I)

Example

Construct the SLR parsing table for G_6.

$$
\begin{array}{llll}
(1) & E & \longrightarrow & E + T \\
(2) & E & \longrightarrow & T \\
(3) & T & \longrightarrow & T * F \\
(4) & T & \longrightarrow & F \\
(5) & F & \longrightarrow & (\, E \,) \\
(6) & F & \longrightarrow & \mathbf{id}
\end{array}
$$

Note:

si means "shift state i".

rj means "reduce rule j".

Exercise (I)

Example

Construct the SLR parsing table for G_6.

$$\begin{array}{llll}
(1) & E & \longrightarrow & E + T \\
(2) & E & \longrightarrow & T \\
(3) & T & \longrightarrow & T * F \\
(4) & T & \longrightarrow & F \\
(5) & F & \longrightarrow & (\ E\ ) \\
(6) & F & \longrightarrow & \textbf{id}
\end{array}$$

Note:

si means "shift state i".

rj means "reduce rule j".

Exercise (II)

Example (table construction , automaton)

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

The LR Parsing Algorithm

- The LR parsing algorithm takes a CFG G and an input string w as input and produces a reduction of w to S, the start variable of G.
- Note that the basic algorithm to be presented is a general LR parser.
- Depending on how the parsing table is constructed, we get special types of LR parsers.
- The algorithm uses a stack together with the parse table to parse the input.

The LR Parsing Algorithm

- The LR parsing algorithm takes a CFG G and an input string w as input and produces a reduction of w to S, the start variable of G.
- Note that the basic algorithm to be presented is a general LR parser.
- Depending on how the parsing table is constructed, we get special types of LR parsers.
- The algorithm uses a stack together with the parse table to parse the input.

The LR Parsing Algorithm

- The LR parsing algorithm takes a CFG G and an input string w as input and produces a reduction of w to S, the start variable of G.
- Note that the basic algorithm to be presented is a general LR parser.
- Depending on how the parsing table is constructed, we get special types of LR parsers.
- The algorithm uses a stack together with the parse table to parse the input.

The LR Parsing Algorithm

- The LR parsing algorithm takes a CFG G and an input string w as input and produces a reduction of w to S, the start variable of G.
- Note that the basic algorithm to be presented is a general LR parser.
- Depending on how the parsing table is constructed, we get special types of LR parsers.
- The algorithm uses a stack together with the parse table to parse the input.

The Algorithm

Given $G = \langle V, \Sigma, R, S \rangle$ and w.

1. Construct M_G and the parsing table for G.
2. Push the start state of M_G onto the stack.
3. While (true) do
    1. $s \longleftarrow$ top of the stack state.
    2. $a \longleftarrow$ first symbol of w.
    3. If ACTION(s, a) = "shift t", then
        1. Push t on top of the stack.
        2. $w \longleftarrow w$ with a removed.
    4. If ACTION(s, a) = "reduce $A \rightarrow \alpha$", then
        1. Pop $|\alpha|$ states off the stack.
        2. $t \longleftarrow$ top of the stack state.
        3. Push GOTO(t, A) onto the stack.
        4. Output the rule $A \rightarrow \alpha$.
    5. If ACTION(s, a) = "accept", then break.
    6. Else call error-recovery routine.

Exercise (I)

Example

Trace the operation of the LR parsing algorithm given G_6 and input **id**$*$**id**$+$**id**.

Exercise (II)

Example (algorithm, automaton, table)

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

© Aho et al. (2007)

Grammars Not SLR

Example

Consider the following grammar G_7:

$$\begin{array}{lcl} S & \longrightarrow & L{=}R \mid R \\ L & \longrightarrow & *R \mid \mathbf{id} \\ R & \longrightarrow & L \end{array}$$

Grammars Not SLR: States

Example

I_0: $S' \to \cdot S$
$S \to \cdot L = R$
$S \to \cdot R$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$
$R \to \cdot L$

I_1: $S' \to S \cdot$

I_2: $S \to L \cdot = R$
$R \to L \cdot$

I_3: $S \to R \cdot$

I_4: $L \to * \cdot R$
$R \to \cdot L$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$

I_5: $L \to \mathbf{id} \cdot$

I_6: $S \to L = \cdot R$
$R \to \cdot L$
$L \to \cdot * R$
$L \to \cdot \mathbf{id}$

I_7: $L \to * R \cdot$

I_8: $R \to L \cdot$

I_9: $S \to L = R \cdot$

© Aho et al. (2007)

Grammars Not SLR: Parsing Table

Example

Consider ACTION$(2, =)$.

- Due to $(S \rightarrow L.{=}R)$, we get "shift 6".
- But $= \in Follow(R)$. (Why?)
- Thus, due to $(R \rightarrow L)$, we get "reduce $R \rightarrow L$".
- Hence, a shift/reduce conflict.

Grammars Not SLR: Parsing Table

Example

Consider ACTION$(2, =)$.

- Due to $(S \rightarrow L.{=}R)$, we get "shift 6".
- But $= \in \mathit{Follow}(R)$. (Why?)
- Thus, due to $(R \rightarrow L)$, we get "reduce $R \rightarrow L$".
- Hence, a shift/reduce conflict.

Grammars Not SLR: Parsing Table

Example

Consider ACTION$(2, =)$.

- Due to $(S \rightarrow L.{=}R)$, we get "shift 6".
- But $= \in \mathit{Follow}(R)$. (Why?)
- Thus, due to $(R \rightarrow L)$, we get "reduce $R \rightarrow L$".
- Hence, a shift/reduce conflict.

Grammars Not SLR: Parsing Table

Example

Consider ACTION$(2, =)$.

- Due to $(S \longrightarrow L.{=}R)$, we get "shift 6".
- But $= \in \mathit{Follow}(R)$. (Why?)
- Thus, due to $(R \longrightarrow L)$, we get "reduce $R \longrightarrow L$".
- Hence, a shift/reduce conflict.