Syntax Analysis (Part 2)

Martin Sulzmann

Bottom-Up Parsing

Idea

- Build right-most derivation.
- Scan input and seek for matching right hand sides.

Terminology

- If $S \rightarrow^* \beta$ then β is a *sentential form* of G.
- *Right sentential form* $=$ right most reduction of non-terminal symbols.
- *Handle* $=$ Right hand side α of a production $A \rightarrow \alpha$ where α is part of a right sentential form $S \Rightarrow^* \beta$.

Example

Consider

$$
\begin{array}{lcl}
S & \rightarrow & aABe \ \ (1) \\
A & \rightarrow & Abc \ \ (2) \mid b \ \ (3) \\
B & \rightarrow & d \ \ (4)
\end{array}
$$

The sentence *abbcde* can be *reduced* to S:

Rule	Sentential form
3	$a\underline{b}bcde$
2	$a\underline{Abc}de$
4	$aA\underline{d}e$
1	$\underline{aABe}$
	S

$$
\left(
\begin{array}{lcl}
S & \Rightarrow & aA\underline{B}e \\
 & \Rightarrow & a\underline{A}de \\
 & \Rightarrow & a\underline{A}bcde \\
 & \Rightarrow & abbcde \\
 & & \\
\multicolumn{3}{l}{\text{right-most derivation}}
\end{array}
\right)
$$

Marked forms are the *handles*.

Another Example

Consider

$$E \quad \rightarrow \quad E + E \mid E * E \mid (E) \mid id$$

Right-most derivation:
$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * id_3 \Rightarrow E + id_2 * id_3 \Rightarrow id_1 + id_2 * id_3$
Consider how the input string is "reduced":

right-sentential form	Handle	reducing production
$id_1 + id_2 * id_3$	id_1	$E \rightarrow id$
$E + id_2 * id_3$	id_2	$E \rightarrow id$
$E + E * id_3$	id_3	$E \rightarrow id$
$E + E * E$	$E * E$	$E \rightarrow E * E$
$E + E$	$E + E$	$E \rightarrow E + E$
E		

Handle Pruning (=Shift/Reduce Parsing)

Goal

Scan input, recognize handles and replace (if match found) by left-hand side until we reach the start symbol.

Shift-Reduce Approach

Make use of stack to keep track of "partial" handles.

- Shift:
  - Seek for handle.
  - "Shift" input symbols on some stack.
- Reduce:
  - Match for handle detected.
  - Replace right hand by left hand side.
  - For $A \to \beta$:
    - "pop" $|\beta|$ symbols
    - "pus" A onto stack

Example

Consider

$$E \quad \rightarrow \quad E + E \mid E * E \mid (E) \mid id$$

Stack	Input	Action
$\$$	$(id * id)\$$	shift
$\$($	$id * id)\$$	shift
$\$(id$	$*id)\$$	reduce $E \rightarrow id$
$\$(E$	$*id)\$$	shift
$\$(E*$	$id)\$$	shift
$\$(E * id$	$)\$$	reduce $E \rightarrow id$
$\$(E * E$	$)\$$	reduce $E \rightarrow E * E$
$\$(E$	$)\$$	shift
$\$(E)$	$\$$	reduce $E \rightarrow (E)$
$\$E$	$\$$	accept

$$E \Rightarrow (E) \Rightarrow (E * E) \Rightarrow (E * id) \Rightarrow (id * id)$$

LR Parsing

Approach

Instance of shift-reduce parser where we employ FSA to recognize handles.

Method

- Build a right-most derivation in reverse $w \Leftarrow^* S$.
- Maintain stack of *viable prefixes*, i.e. forms α such that $S \Rightarrow^* \alpha w$ for some $w \in V_t^*$.
- Shift action: Move token from input w to stack.
- Reduce action: Apply grammar rule (in reverse) to top section of stack.

Skeleton LR Parser

Algorithm

```
push s0
token:= next_token()
repeat
  s:= top of stack
  if action[s,token] = ``shift si'' then
    push si
    token:= next_token()
  else if action[s,token] = ``reduce A->beta'' then
        pop |beta| states
        s':= top of stack
        push goto[s',A]
  else if action[s,token] = ``accept'' then done
  else error
```

Example

$$
\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & T + E \mid T \\
T & \rightarrow & F * T \mid F \\
F & \rightarrow & id
\end{array}
$$

state	*action*				*goto*		
	id	+	*	$	*E*	*T*	*F*
0	*s4*	-	-	-	1	2	3
1	-	-	-	*acc*	-	-	-
2	-	*s5*	-	*r3*	-	-	-
3	-	*r5*	*s6*	*r5*	-	-	-
4	-	*r6*	*r6*	*r6*	-	-	-
5	*s4*	-	-	-	7	2	3
6	*s4*	-	-	-	-	8	3
7	-	-	-	*r2*	-	-	-
8	-	*r4*	-	*r4*	-	-	-

Stack	Input	Action
$ 0	$id * id + id\$$	*s4*
$ 0 4	$*id + id\$$	*r6*
$ 0 3	$*id + id\$$	*s6*
$ 0 3 6	$id + id\$$	*s4*
$ 0 3 6 4	$+id\$$	*r6*
$ 0 3 6 3	$+id\$$	*r5*
$ 0 3 6 8	$+id\$$	*r4*
$ 0 2	$+id\$$	*s5*
$ 0 2 5	$id\$$	*s4*
$ 0 2 5 4	$	*r6*
$ 0 2 5 3	$	*r5*
$ 0 2 5 2	$	*r3*
$ 0 2 5 7	$	*r2*
$ 0 1	$	*acc*

LR(k) Grammars

Idea

A grammar G is LR(k) if given a right-most derivation

$$S = \gamma_0 \Rightarrow_R \gamma_1 \Rightarrow_R \ldots \Rightarrow_R \gamma_n = w$$

we can, for each right-sentential form in the derivation:

1. isolate the handle of each right-sentential form, and
2. determine the production by which to reduce

by scanning γ_i from left to right, going at most k symbols beyond the right end of the handle of γ_i.

LR(k) Items

LR(k) Items

The table construction algorithm uses sets of LR(k) items or *configurations* to represent the possible states in a parse.

A LR(k) item is a pair $[\alpha, \beta]$ where

- α is a production from G with a $\bullet$ at some position in the rhs, marking how much of the rhs of a production has already been seen
- β is a lookahead string containing k symbols (terminals or $)

As we will see, the two cases of interest are k=0 and k=1.

Example

The $\bullet$ indicates how much of an item we have seen at a given state in the parse:

$[A \to \bullet XYZ]$ indicates that the parser is looking for a string that can be derived from XYZ.

$[A \to XY \bullet Z]$ indicates that the parser has seen a string derived from XY and is looking for one derivable from Z.

LR(0) items (no lookahead):

$A \to XYZ$ generates 4 LR(0) items:

1. $[A \to \bullet XYZ]$
2. $[A \to X \bullet YZ]$
3. $[A \to XY \bullet Z]$
4. $[A \to XYZ \bullet]$

The CFSM

The Characteristic Finite State Machine (CFSM) for a grammar is a DFA which recognizes *viable prefixes* of right-sentential forms (Recall: A viable prefix is any prefix that does not extend beyond the handle).

It accepts when a handle has been discovered and needs to be reduced.

To construct the CFSM we need two functions:

- closure0(I) to build its states
- goto0(I,X) to determine its transitions

closure0

Given an item $[A \to \alpha \bullet B\beta]$, its closure contains the item itself and any other item that can generate legal substrings to follow α.

Thus, if the parser has viable prefix α on the stack, the input should reduce to $B\beta$ (or γ for other item $[B \to \bullet\gamma]$ in the closure).

```
closure0(I):
      repeat
            if [A → α • Bβ] ∈ I and B → γ is a rule
            then add [B → •γ] to I
      until no more items can be added to I
      return I
```

goto0

goto0(I,X) is the closure of the set of all items $[A \to \alpha X \bullet \beta]$ such that $[A \to \alpha \bullet X\beta] \in I$.

If I is the set of valid items fro some viable prefix γ, then goto0(I,X) is the set of valid items for the viable prefix γX.

goto0(I,X) represents the state after recognizing X in state I.

goto0(I,X):
    let J be the set of items $[A \to \alpha X \bullet \beta]$
    such that $[A \to \alpha \bullet X\beta] \in I$
    return closure0(J)

Example

Consider

$$\begin{array}{lcl} P & \to & CP \mid \epsilon \\ C & \to & agBe \mid ae \\ B & \to & acB \mid a \end{array}$$

Assume $I = \{[C \to ag \bullet Be]\}$. Then (I omit [and] for simplicity)

$$closure0(I) \quad = \quad \left\{ \begin{array}{l} C \to ag \bullet Be, \\ B \to \bullet acB, \\ B \to \bullet a \end{array} \right\}$$

Assume $I = \{[P \to \bullet], [P \to \bullet CP]\}$. Then

$$goto0(I, C) \quad = \quad \left\{ \begin{array}{l} P \to C \bullet P, \\ P \to \bullet CP, \\ P \to \bullet, \\ C \to \bullet agBe, \\ C \to \bullet ae \end{array} \right\}$$

Constructing the LR(0) Item Sets

We introduce a new start symbol S' where $S' \to S$ (some presentations assume $S' \to S\$$.)

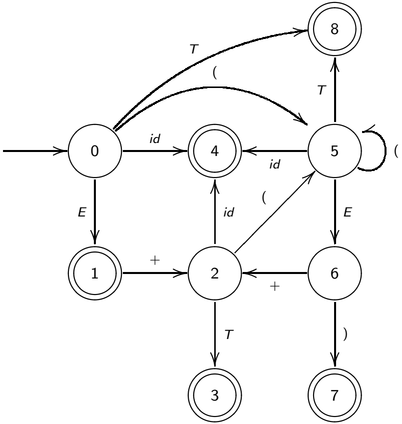
$States$ = $\{closure0(\{S' \to \bullet S\})\}$
`while we can keep making changes`
    `for each` $I \in States$ `and symbol` X
    `I' = goto0(I,X)`
    `if` $I \notin States$ `then` $States$ = $States \cup \{I'\}$

LR(0) Example

Grammar:

$$
\begin{array}{lcl}
S & \to & E \ (1) \\
E & \to & E + T \ (2) \mid T \ (3) \\
T & \to & id \ (4) \mid (E) \ (5)
\end{array}
$$

CFSM:



LR(0) items:

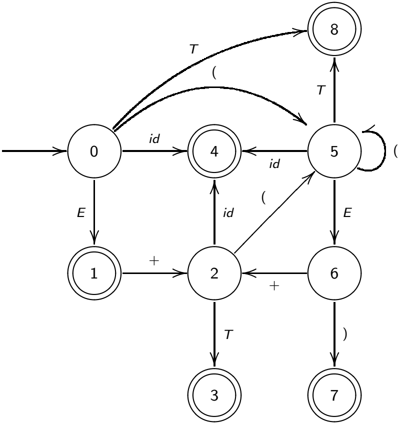
$$
\begin{array}{lll}
I_0: & S \to \bullet E \\
 & E \to \bullet E + T \\
 & E \to \bullet T \\
 & T \to \bullet id \\
 & T \to \bullet (E) \\
I_1: & S \to E \bullet \\
 & E \to E \bullet + T \\
I_2: & E \to E + \bullet T \\
 & T \to \bullet id \\
 & T \to \bullet (E) \\
I_3: & E \to E + T \bullet \\
I_4: & T \to id \bullet \\
I_5: & T \to (\bullet E) \\
 & E \to \bullet E + T \\
 & E \to \bullet T \\
 & T \to \bullet id \\
 & T \to \bullet (E) \\
I_6: & T \to (E \bullet) \\
 & E \to E \bullet + T \\
I_7: & T \to (E) \bullet \\
I_8: & E \to T \bullet
\end{array}
$$

Constructing the Parsing Table

We have constructed LR(0) items and CFSM (state i of the CFSM is constructed from I_i).

- If $A \to \alpha \bullet a\beta \in I_i$ and goto0(I_i,a)=I_j then action(i,a)=shift j.
- If $A \to \alpha\bullet \in I$ then action(I,a) $=$ reduce($A \to \alpha$) for all a.
- If $S' \to S\bullet \in I$ then action(I,\$) $=$ accept.
- If goto0(I_i,A) $=$ I_j (where $A \in V_n$) then goto(i,A)=j.
- Set undefined entries in action and goto to "error".
- Set initial state of parser to closure0([$S' \to \bullet S$]).

LR(0) Example



state	action					goto		
	id	(	)	+	\$	*S*	*E*	*T*
0	*s*4	*s*5					1	8
1					*acc*			
2	*s*4	*s*5						3
3	*r*2	*r*2	*r*2	*r*2	*r*2			
4	*r*4	*r*4	*r*4	*r*4	*r*4			
5	*s*4	*s*5					6	8
6			*s*7	*s*2				
7	*r*5	*r*5	*r*5	*r*5	*r*5			
8	*r*3	*r*3	*r*3	*r*3	*r*3			

Another Example

Consider

$$
\begin{array}{lcl}
S & \rightarrow & A \mid Bc \mid D \\
A & \rightarrow & a \mid b \\
B & \rightarrow & a \\
D & \rightarrow & bc
\end{array}
$$

Is this grammar LR(0)?

Another Example (2)

Grammar:

$$
\begin{array}{lcl}
S & \rightarrow & A \mid Bc \mid D \\
A & \rightarrow & a \mid b \\
B & \rightarrow & a \\
D & \rightarrow & bc
\end{array}
$$

LR(0) items:

$$
\begin{array}{ll}
I_0 : & S' \rightarrow \bullet S \\
 & S \rightarrow \bullet A \\
 & S \rightarrow \bullet Bc \\
 & S \rightarrow \bullet D \\
 & A \rightarrow \bullet a \\
 & A \rightarrow \bullet b \\
 & B \rightarrow \bullet a \\
 & D \rightarrow \bullet bc \\
I_1 : & S' \rightarrow S \bullet \\
I_2 : & S \rightarrow A \bullet
\end{array}
\qquad
\begin{array}{ll}
I_3 : & S \rightarrow B \bullet c \\
I_4 : & S \rightarrow Bc \bullet \\
I_5 : & S \rightarrow D \bullet \\
I_6 : & A \rightarrow a \bullet \\
 & B \rightarrow a \bullet \\
I_7 : & A \rightarrow b \bullet \\
 & D \rightarrow b \bullet c \\
 & \ldots
\end{array}
$$

Another Example (3)

Grammar:

$$\begin{array}{lcl} S & \rightarrow & A \mid Bc \mid D \\ A & \rightarrow & a \mid b \\ B & \rightarrow & a \\ D & \rightarrow & bc \end{array}$$

LR(0) items:

$$\begin{array}{ll} I_0: & S' \rightarrow \bullet S \\ & S \rightarrow \bullet A \\ & S \rightarrow \bullet Bc \\ & S \rightarrow \bullet D \\ & A \rightarrow \bullet a \\ & A \rightarrow \bullet b \\ & B \rightarrow \bullet a \\ & D \rightarrow \bullet bc \\ I_1: & S' \rightarrow S\bullet \\ I_2: & S \rightarrow A\bullet \end{array}$$

$$\begin{array}{lll} I_3: & S \rightarrow B \bullet c & \\ I_4: & S \rightarrow Bc\bullet & \\ I_5: & S \rightarrow D\bullet & \\ I_6: & A \rightarrow a\bullet & \text{reduce} \\ & B \rightarrow a\bullet & \text{reduce conflict!} \\ I_7: & A \rightarrow b\bullet & \\ & D \rightarrow b \bullet c & \\ & \ldots & \end{array}$$

Another Example (4)

Grammar:

$$\begin{array}{lcl} S & \rightarrow & A \mid Bc \mid D \\ A & \rightarrow & a \mid b \\ B & \rightarrow & a \\ D & \rightarrow & bc \end{array}$$

LR(0) items:

$$\begin{array}{lll} I_0: & S' \rightarrow \bullet S & \\ & S \rightarrow \bullet A & \\ & S \rightarrow \bullet Bc & \\ & S \rightarrow \bullet D & \\ & A \rightarrow \bullet a & \\ & A \rightarrow \bullet b & \\ & B \rightarrow \bullet a & \\ & D \rightarrow \bullet bc & \\ I_1: & S' \rightarrow S \bullet & \\ I_2: & S \rightarrow A \bullet & \\ I_3: & S \rightarrow B \bullet c & \\ I_4: & S \rightarrow Bc \bullet & \\ I_5: & S \rightarrow D \bullet & \\ I_6: & A \rightarrow a \bullet & \text{reduce} \\ & B \rightarrow a \bullet & \text{reduce conflict!} \\ I_7: & A \rightarrow b \bullet & \text{reduce} \\ & D \rightarrow b \bullet c & \text{shift conflict!} \\ & \ldots & \end{array}$$

Above grammar is not LR(0).

LR Parsing

Short Summary

If LR(0) parsing tables contains multiply-defined action entries then G is not LR(0).

There are two possible types of conflicts:

- shift-reduce
- reduce-reduce

Conflicts can be resolved by *looking ahead*.

SLR(1): Simple Lookahead LR

SLR(1)

Add lookaheads after building LR(0) item sets. We construct the SLR(1) parsing table as follows:

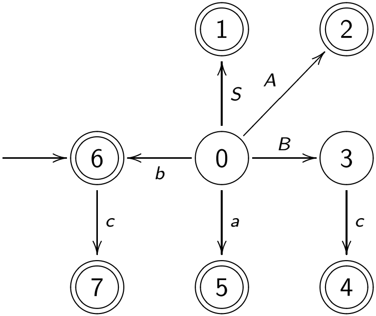
- If $A \to \alpha \bullet a\beta \in I$ ($a \in V_t$) and goto0(I,a)=I_i then action(I,a)=shift i.
- If $A \to \alpha\bullet \in I$ then action(I,a) = reduce($A \to \alpha$) for all $a \in Follow(A)$.
- If $S' \to S\bullet \in I$ then action(I,\$) = accept.
- If goto0(I_i,A) = I_j (where $A \in V_n$) then goto(i,A)=j.
- Set undefined entries in action and goto to “error”.
- Set initial state of parser to closure0([$S' \to \bullet S$]).

SLR(1) but not LR(0) Example

$$
\begin{array}{lcl}
S & \rightarrow & A\ (1) \mid Bc\ (2) \mid bc\ (3) \\
A & \rightarrow & a\ (4) \mid b\ (5) \\
B & \rightarrow & a\ (6)
\end{array}
$$

$$
\begin{array}{l}
Follow(S) = \{\$\} \\
Follow(A) = \{\$\} \\
Follow(B) = \{c\}
\end{array}
$$

$$
\begin{array}{lllll}
I_0: & S' \rightarrow \bullet S & \quad & I_2 & S \rightarrow A\bullet \\
 & S \rightarrow \bullet A & & I_3 & S \rightarrow B \bullet c \\
 & S \rightarrow \bullet Bc & & I_4 & S \rightarrow Bc\bullet \\
 & S \rightarrow \bullet bc & & I_5 & A \rightarrow a\bullet \\
 & A \rightarrow \bullet a & & & B \rightarrow a\bullet \\
 & A \rightarrow \bullet b & & I_6 & A \rightarrow b\bullet \\
 & B \rightarrow \bullet a & & & S \rightarrow b \bullet c \\
I_1: & S' \rightarrow S\bullet & & I_7 & S \rightarrow Bc\bullet
\end{array}
$$



SLR(1) Parsing Table

$$\begin{array}{lcl} S & \rightarrow & A\ (1) \mid Bc\ (2) \mid bc\ (3) \\ A & \rightarrow & a\ (4) \mid b\ (5) \\ B & \rightarrow & a\ (6) \end{array}$$

$$\begin{array}{l} Follow(S) = \{\$\} \\ Follow(A) = \{\$\} \\ Follow(B) = \{c\} \end{array}$$

state	*action*				*goto*		
	a	*b*	*c*	\$	*S*	*A*	*B*
0	*s*5	*s*6			1	2	3
1				*acc*			
2				*r*1			
3			*s*4				
4				*r*2			
5			*r*6	*r*4			
6			*s*7				
7				*r*2			

Limitations of SLR(1)

Consider

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

LR(0) items:

$$\begin{array}{llllll} I_0 : & S' \rightarrow \bullet S & I_1 : & S' \rightarrow S \bullet & & \\ & S \rightarrow \bullet L = R & I_2 : & S \rightarrow L \bullet = R & \text{shift!} & \\ & S \rightarrow \bullet R & & R \rightarrow L \bullet & \text{reduce!} & (= \in Follow(R)) \\ & L \rightarrow \bullet * R & & & & \\ & L \rightarrow \bullet id & & & & \\ & R \rightarrow \bullet id & & & & \end{array}$$

Limitations of SLR(1) (cont'd)

Examples shows that it can be insufficient to consider *Follow* sets.

Assume we are parsing $id = id$.

In item 2, the shift action is the only correct choice (to reduce L to R gets us nowhere).

Note that item $R \rightarrow L\bullet$ came via $S' \Rightarrow S \Rightarrow R \Rightarrow L$ (each symbol can only be "followed" by \$).

Hence, we need to make an effort to remember the actual right context.

LR(1) Items Revisited

Idea

We can incorporate the extra right-context information into items.

An LR(1) item is of the form $[A \to \alpha \bullet \beta, t]$ where $t \in V_t \cup \{\$\}$.

An item $[A \to \alpha \bullet, a]$ calls for the reduction $A \to \alpha$ only if the next input symbol is a.

LR(1) Items Revisited (cont'd)

closure1 with Context

closure1(I):

  while new items can be added

   if $[A \to \alpha \bullet B\gamma, c] \in I$ and $B \to \delta$ is a rule

   then for each $b \in \mathit{First}(\gamma c)$

    add $[B \to \bullet\delta, b]$ to I

goto1 with Context

goto1(I,X):

  let J be the set of items $[A \to \alpha X \bullet \beta, a]$

  such that $[A \to \alpha \bullet X\beta, a] \in I$

  return closure0(J)

Initial State

$[S' \to \bullet S, \$]$

LR(1) Parsing Table

Algorithm

For each set I_i of LR(1) items:

- If $[A \to \alpha \bullet a\beta] \in I_i$ and goto1(I_i,a)=I_j then action(i,a)=shift j.
- If $[A \to \alpha\bullet, b] \in I_i$ $(A \neq S')$ then action(i,b)= reduce($A \to \alpha$).
- If $[S' \to S\bullet, \$] \in I_i$ then action(i,\$)= accept.
- If goto1(I_i,A) = I_j (where $A \in V_n$) then goto(i,A)=j.

All remaining entries are set to “error”.

Initial state contains $[S' \to \bullet S, \$]$.

Recall Example

Consider

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

LR(1) items:

$$
\begin{array}{ll}
I_0 : & [S' \rightarrow \bullet S, \$] \\
& [S \rightarrow \bullet L = R, \$] \\
& [S \rightarrow \bullet R, \$] \\
& [L \rightarrow \bullet * R, =] \\
& [L \rightarrow \bullet id, =] \\
& [R \rightarrow \bullet L, \$]
\end{array}
\qquad
\begin{array}{ll}
I_1 : & [S' \rightarrow S \bullet, \$] \\
I_2 : & [S \rightarrow L \bullet = R, \$] \ \boxed{\text{no } =} \\
& [R \rightarrow L \bullet, \$]
\end{array}
$$

$\boxed{= \text{ due to } First(= R)}$

Another Example

Grammar:

$$
\begin{array}{lcl}
S' & \rightarrow & S\ (0) \\
S & \rightarrow & CC\ (1) \\
C & \rightarrow & cC\ (2) \mid d\ (3)
\end{array}
$$

state	*action*			*goto*	
	c	*d*	$	*S*	*C*
0	*s*3	*s*4		1	2
1			*acc*		
2	*s*6	*s*7			5
3	*s*3	*s*4			8
4	*r*3	*r*3			
5			*r*1		
6	*s*6	*s*7			9
7			*r*3		
8	*r*2	*r*2			
9			*r*2		

LR(1) items:

$$
\begin{array}{lll}
I_0: & S' \rightarrow \bullet S, \$ \\
 & S \rightarrow \bullet CC, \$ \\
 & C \rightarrow \bullet cC, c \mid d \\
 & C \rightarrow \bullet d, c \mid d \\
I_1: & S' \rightarrow S\bullet, \$ \\
I_2: & S \rightarrow C \bullet C, \$ \\
 & C \rightarrow \bullet cC, \$ \\
 & C \rightarrow \bullet d, \$ \\
I_3: & C \rightarrow c \bullet C, c \mid d \\
 & C \rightarrow \bullet cC, c \mid d \\
 & C \rightarrow \bullet d, c \mid d
\end{array}
\qquad
\begin{array}{lll}
I_4: & C \rightarrow d\bullet, c \mid d \\
I_5: & S \rightarrow CC\bullet, \$ \\
I_6: & C \rightarrow c \bullet C, \$ \\
 & C \rightarrow \bullet cC, \$ \\
 & C \rightarrow \bullet d, \$ \\
I_7: & C \rightarrow d\bullet, \$ \\
I_8: & C \rightarrow cC\bullet, c \mid d \\
I_9: & C \rightarrow cC\bullet, \$
\end{array}
$$

Another Example (2)

Grammar:

$$\begin{array}{lcl} S' & \rightarrow & S\ (0) \\ S & \rightarrow & CC\ (1) \\ C & \rightarrow & cC\ (2) \mid d\ (3) \end{array}$$

state	*action*			*goto*	
	c	*d*	$	*S*	*C*
0	*s*3	*s*4		1	2
1			*acc*		
2	*s*6	*s*7			5
3	*s*3	*s*4			8
4	*r*3	*r*3			
5			*r*1		
6	*s*6	*s*7			9
7			*r*3		
8	*r*2	*r*2			
9			*r*2		

LR(1) items:

$$\begin{array}{lll} I_0: & S' \rightarrow \bullet S, \$ \\ & S \rightarrow \bullet CC, \$ \\ & C \rightarrow \bullet cC, c \mid d \\ & C \rightarrow \bullet d, c \mid d \\ I_1: & S' \rightarrow S\bullet, \$ \\ I_2: & S \rightarrow C \bullet C, \$ \\ & C \rightarrow \bullet cC, \$ \\ & C \rightarrow \bullet d, \$ \\ I_3: & C \rightarrow c \bullet C, c \mid d \\ & C \rightarrow \bullet cC, c \mid d \\ & C \rightarrow \bullet d, c \mid d \end{array}$$

$$\begin{array}{ll} I_4: & C \rightarrow d\bullet, c \mid d \\ I_5: & S \rightarrow CC\bullet, \$ \\ I_6: & C \rightarrow c \bullet C, \$ \\ & C \rightarrow \bullet cC, \$ \\ & C \rightarrow \bullet d, \$ \\ I_7: & C \rightarrow d\bullet, \$ \\ I_8: & C \rightarrow cC\bullet, c \mid d \\ I_9: & C \rightarrow cC\bullet, \$ \end{array}$$

I_3 and I_6 have the same "core"!

LALR: Lookahead LR

Idea

Define the *core* of a set of LR(1) items to be the set of LR(0) items derived by ignoring the lookahead symbols.

E.g. the two sets

- $\{[C \to c \bullet C, c \mid d], [C \to \bullet cC, c \mid d], [C \to \bullet d, c \mid d]\}$
- $\{[C \to c \bullet C, \$], [C \to \bullet cC, \$], [C \to \bullet d, \$]\}$

have the same core.

Key idea: If two sets of LR(1) items, I_i and I_j, have the same core, we can merge the states that represent them in the action and goto tables.

LALR(1) algorithm

Algorithm

- Construct LR(1) items.
- Find all cores among sets of LR(1) items, replace these sets by their union, update goto function (incrementally).
- For each I_i
  - If $[A \to \alpha \bullet a\beta, b] \in I_i$ and goto1(I_i,a)=I_j then action(i,a)=shift j.
  - If $[A \to \alpha\bullet, b] \in I_i$ $(A \neq S')$ then action(i,b)= reduce($A \to \alpha$).
  - If $[S' \to S\bullet, \$] \in I_i$ then action(i,\$)= accept.
- If goto1(I_i,A) $=$ I_j (where $A \in V_n$) then goto(i,A)=j.

All remaining entries are set to "error".
Initial state $=$ $closure1(\{[S' \to \bullet S, \$]\})$.
More efficient LALR construction possible, see textbook.

Recall Example

Grammar:

$$
\begin{array}{lll}
S' & \rightarrow & S\ (0) \\
S & \rightarrow & CC\ (1) \\
C & \rightarrow & cC\ (2) \mid d\ (3)
\end{array}
$$

state	action			goto	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LR(1) items:

$$
\begin{array}{lll}
I_0: & S' \rightarrow \bullet S, \$ \\
 & S \rightarrow \bullet CC, \$ \\
 & C \rightarrow \bullet cC, c \mid d \\
 & C \rightarrow \bullet d, c \mid d \\
I_1: & S' \rightarrow S\bullet, \$ \\
I_2: & S \rightarrow C \bullet C, \$ \\
 & C \rightarrow \bullet cC, \$ \\
 & C \rightarrow \bullet d, \$ \\
I_3: & C \rightarrow c \bullet C, c \mid d \\
 & C \rightarrow \bullet cC, c \mid d \\
 & C \rightarrow \bullet d, c \mid d
\end{array}
$$

$$
\begin{array}{lll}
I_4: & C \rightarrow d\bullet, c \mid d \\
I_5: & S \rightarrow CC\bullet, \$ \\
I_6: & C \rightarrow c \bullet C, \$ \\
 & C \rightarrow \bullet cC, \$ \\
 & C \rightarrow \bullet d, \$ \\
I_7: & C \rightarrow d\bullet, \$ \\
I_8: & C \rightarrow cC\bullet, c \mid d \\
I_9: & C \rightarrow cC\bullet, \$
\end{array}
$$

Recall Example (2)

Grammar:

$$\begin{array}{lcl} S' & \rightarrow & S\ (0) \\ S & \rightarrow & CC\ (1) \\ C & \rightarrow & cC\ (2) \mid d\ (3) \end{array}$$

state	*action*			*goto*	
	c	d	\$	S	C
0	$s3$	$s4$		1	2
1			*acc*		
2	$s6$	$s7$			5
3	$s3$	$s4$			8
4	$r3$	$r3$			
5			$r1$		
6	$s6$	$s7$			9
7			$r3$		
8	$r2$	$r2$			
9			$r2$		

LR(1) items:

$$\begin{array}{ll} I_0: & S' \rightarrow \bullet S, \$ \\ & S \rightarrow \bullet CC, \$ \\ & C \rightarrow \bullet cC, c \mid d \\ & C \rightarrow \bullet d, c \mid d \\ I_1: & S' \rightarrow S\bullet, \$ \\ I_2: & S \rightarrow C \bullet C, \$ \\ & C \rightarrow \bullet cC, \$ \\ & C \rightarrow \bullet d, \$ \\ I_3: & C \rightarrow c \bullet C, c \mid d \\ & C \rightarrow \bullet cC, c \mid d \\ & C \rightarrow \bullet d, c \mid d \end{array}$$

$$\begin{array}{ll} I_4: & C \rightarrow d\bullet, c \mid d \\ I_5: & S \rightarrow CC\bullet, \$ \\ I_6: & C \rightarrow c \bullet C, \$ \\ & C \rightarrow \bullet cC, \$ \\ & C \rightarrow \bullet d, \$ \\ I_7: & C \rightarrow d\bullet, \$ \\ I_8: & C \rightarrow cC\bullet, c \mid d \\ I_9: & C \rightarrow cC\bullet, \$ \end{array}$$

Merged states:

$$\begin{array}{ll} I_{36}: & C \rightarrow c \bullet C, c \mid d \mid \$ \\ & C \rightarrow \bullet cC, c \mid d \mid \$ \\ & C \rightarrow \bullet d, c \mid d \mid \$ \\ I_{47}: & C \rightarrow d\bullet, c \mid d \mid \$ \\ I_{89}: & C \rightarrow cC\bullet, c \mid d \mid \$ \end{array}$$

Recall Example (3)

Grammar:

$$\begin{array}{lcl} S' & \rightarrow & S\ (0) \\ S & \rightarrow & CC\ (1) \\ C & \rightarrow & cC\ (2) \mid d\ (3) \end{array}$$

Updated table:

state	*action*			*goto*	
	c	d	\$	S	C
0	$s36$	$s47$		1	2
1			acc		
2	$s36$	$s47$			5
36	$s36$	$s47$			89
47	$r3$	$r3$	$r3$		
5			$r1$		
89	$r2$	$r2$	$r2$		

LR(1) items:

$$\begin{array}{lllll} I_0: & S' \rightarrow \bullet S, \$ & & I_4: & C \rightarrow d\bullet, c \mid d \\ & S \rightarrow \bullet CC, \$ & & I_5: & S \rightarrow CC\bullet, \$ \\ & C \rightarrow \bullet cC, c \mid d & & I_6: & C \rightarrow c \bullet C, \$ \\ & C \rightarrow \bullet d, c \mid d & & & C \rightarrow \bullet cC, \$ \\ I_1: & S' \rightarrow S\bullet, \$ & & & C \rightarrow \bullet d, \$ \\ I_2: & S \rightarrow C \bullet C, \$ & & I_7: & C \rightarrow d\bullet, \$ \\ & C \rightarrow \bullet cC, \$ & & I_8: & C \rightarrow cC\bullet, c \mid d \\ & C \rightarrow \bullet d, \$ & & I_9: & C \rightarrow cC\bullet, \$ \\ I_3: & C \rightarrow c \bullet C, c \mid d & & & \\ & C \rightarrow \bullet cC, c \mid d & & & \\ & C \rightarrow \bullet d, c \mid d & & & \end{array}$$

Merged states:

$$\begin{array}{ll} I_{36}: & C \rightarrow c \bullet C, c \mid d \mid \$ \\ & C \rightarrow \bullet cC, c \mid d \mid \$ \\ & C \rightarrow \bullet d, c \mid d \mid \$ \\ I_{47}: & C \rightarrow d\bullet, c \mid d \mid \$ \\ I_{89}: & C \rightarrow cC\bullet, c \mid d \mid \$ \end{array}$$

Summary

- LR more powerful than LL.
- ANTLR is a "pretty cool" Java-based LL parser.
- LALR good enough in practice (see yacc).
- Precedence can also be used to resolve shift/reduce conflicts.
  E.g. higher precedence $\Rightarrow$ shift, left associative $\Rightarrow$ reduce.
- Error recovery important (but no time!), please consult text book.