LR parsing

- LR(k): deterministic bottom-up parsing using rightmost derivations with k-symbol lookahead.

eg: non-determinism in bottom-up parse

$$
\begin{aligned}
S &\longrightarrow \; aAb \mid BaAa \\
A &\longrightarrow \; ab \mid b \\
B &\longrightarrow \; Bb \mid b
\end{aligned}
$$

input: $aabb \Rightarrow aAb \qquad aaAb \qquad aaBb$?

- Towards determinism in BUP: How to locate what to reduce, and how to decide by which rule to do the reduction.

  Recall that bottom-up shift-reduce parsers work on *right* sentential forms; they obtain these forms in reverse order.

- *handle*: use of a rule $A \to \beta$ at certain position in a right sentential form.

  $S \overset{R}{\Rightarrow}_* \alpha A w \overset{R}{\Rightarrow} \alpha \beta w$

  ($A \to \beta$ is a handle at position after α; $w \in \Sigma^*$).

- There may be more than one handle if the grammar is ambiguous.

$$
\begin{aligned}
S &\rightarrow\ A \\
A &\rightarrow\ T \mid A + T \\
T &\rightarrow\ b \mid (A)
\end{aligned}
$$

```
input b+(b+b)       (T->b at positions 1,4,6)
      T+(b+b)       (A->T at 0; T->b at 4,6)
      A+(T+b)       (not S->A at 1; A->T at 4; T->b at 6)
      A+(A+b)       (S->A at 4; T->b at 6; not S->A at 1)
      A+(A+T)       (A->A+T at 4; A->T at 6; not S->A  1,4)
      A+(A)         (S->A at 4; not S->A at 1; T->(A) at 3)
      A+T           (A->T at 3; not S->A at 1; A->A+T at 1)
      A
      S
```

The ones that can lead to S are handles.

- Bottom-up parsing can be seen as ’handle pruning’:

  Start with input

  locate a handle

  use handle to reduce the sentential form

  do until S is reached

- Problems: how to locate the handle; reduce by which rule?

  Using a stack to keep sentential forms solves the first problem: the right end of the handle is always on top of the stack. The left end must be found.

Knowledge of context in which a rule can lead to a viable alternative solves the second problem: don't use the rule if it's context is not satisfied.

- *viable prefix (LR(0) context)*: the set of prefixes of a right-sentential form that can appear on stack.

$$S \overset{R}{\Rightarrow}_* uAv \overset{A \rightarrow w}{\Rightarrow} uwv \qquad v \in \Sigma^*$$

uw is a viable prefix

rightmost derivations that terminate with the application of the rule.

Why LR(0) context: no look into v

- Viable prefixes may contain patterns if there's recursion in the grammar. Finding them on-the-fly is costly.

$$\begin{aligned} S \longrightarrow &\quad aA \mid bB \\ A \longrightarrow &\quad abA \mid bB \\ B \longrightarrow &\quad bBc \mid bc \end{aligned}$$

$$S \Rightarrow aA \Rightarrow^i_* a(ab)^i A \Rightarrow a(ab)^i bB \Rightarrow^j_* a(ab)^i bb^j Bc^j \Rightarrow a(ab)^i bb^{j+1} c^{j+1}$$

$$S \Rightarrow bB \Rightarrow^j_* bb^j Bc^j \Rightarrow bb^{j+1} c^{j+1}$$

LR0C($S \rightarrow aA$) $= \{aA\}$

LR0C($A \rightarrow abA$) $= \{a(ab)^i A \mid i > 0\}$

LR0C(B$\rightarrow bBc$) $= \{a(ab)^i b^{j+1} Bc, b^{j+1} Bc \mid i \geq 0, j > 0\}$

why not c^j at the end? Viable prefix includes up to and including RHS of the rule.

- Solutions to the viable prefix problem:

1. Shift-reduce parsing with stack search to locate the left-end of the handle.

2. Shift-reduce parsing with no stack search; design a recognizer to keep progress over the RHSs so that we can tell whether the handle is a viable prefix for a particular rule $\Rightarrow$LR parsing

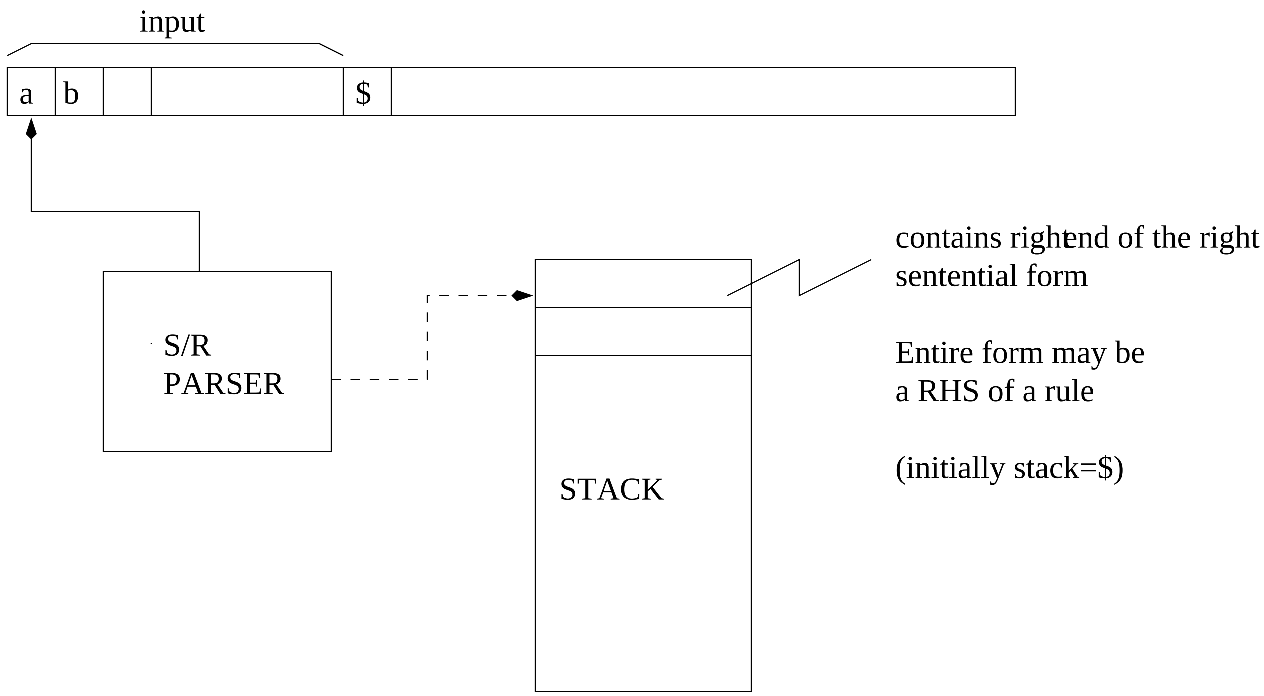
In the first alternative, the stack contains right sentential forms.

In the second alternative, the stack contains right sentential forms and states which give a summary of viable prefixes. No search in stack.

- $S \overset{R}{\Rightarrow}_* uAv \overset{R}{\Rightarrow} uwv \qquad v \in \Sigma^*$

  $A \rightarrow w$ is a handle at the end of u

  uw is the viable prefix (LR0C).



Only viable prefixes appear in the stack

- S/R parsing:

repeat

if a handle is on top, reduce by the handle (pop)

otherwise shift (put in stack)

until accept or input exhausts

$$
G_{AE}: \quad \begin{aligned} S' &\to S\$ \\ S &\to A \\ A &\to T \mid A + T \\ T &\to b \mid (A) \end{aligned}
$$

stack	input	action
$	b+(b+b)$	sh
$b	+(b+b)$	red by T->b

$T		r A->T
$A		sh
$A+	(b+b)$	sh
$A+(	b+b)$	sh
$A+(b	+b)$	r T->b
$A+(T		r A->T
$A+(A		sh
$A+(A+	b)$	sh
$A+(A+b	)$	T->b
$A+(A+T		A->A+T (conflict; but A->T is not a handle)
$A+(A		sh
$A+(A)	$	r T->(A)
$A+T		r A->A+T (conflict)
$A		r S->A (shift-red conflict)
$S		sh
S	-	r S’->S$
$S’		accept

A+A would not be a handle

- Efficiency of left-recursion vs. right-recursion in S/R parsing

```
Xs -> Xs , X | X      vs.    Xs -> X , Xs | X
X  -> id                     X  -> id

input: id1,id2,id3
```

- Reduce-reduce conflict shows the significance of the viable prefix problem. How to choose the right one?

  Shift-reduce conflict shows the significance of handle manipulation and structure of derivations. If the handle is not reduced, can we still get a derivation?

  eg.

```
E -> E * E | E + E | id

stack    input          action

$        id1+id2*id3    sh
$id1     +id2*id3       r E->id1
$E                      sh
$E+      id2*id3        sh
$E+id2   *id3           r E->id2
$E+E                    conflict: shift or reduce?
```

- If shift-reduce conflict is not resolved by grammar re-writing, shift seems to work better for PL grammars.

```
S-> if E then S | if E then S else S

input: if b then if S1 then S2 else S3
```

```
stack              action

$                  sh
..
$if E .. S2        ?
```

- Shift will favor innermost attachment of ’else’; reduce, outermost.

- Does the presence of conflicts mean non-LRness? not necessarily. If the grammar is ambiguous, it can’t be LR(k) for any k. But if the conflict is local, it may still be LR.

- Going from S/R parsing to LR parsing: keep track of ”progress” over the RHS of rules to select the right handle without a search for the left-end of the handle.

- no look beyond the RHS $\Rightarrow$Simple LR (SLR)

  lookahead $\Rightarrow$LR(k) parsing

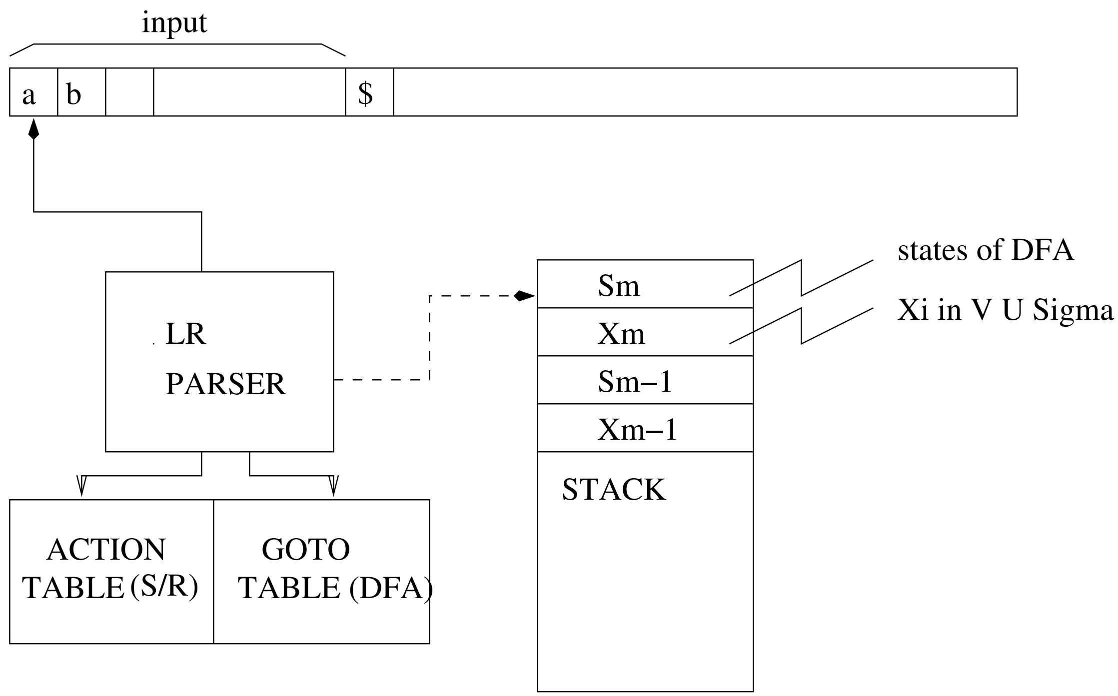
- We'll see that SLR is more informative than simple LL.

- Simple LR: use of LR(0) items (no look past the RHS).

- construct a DFA for recognizing the progress over the RHSs

  eg. $A \rightarrow \cdot \underbrace{B \cdot} \underbrace{C \cdot} \underbrace{a \cdot} \underbrace{D \cdot}$

  $C \rightarrow \cdot \underbrace{B \cdot} \underbrace{C \cdot}$



SHIFT actions and GOTO transitions define the DFA for viable prefixes

SLR=SLR(1)   ACTION/GOTO tables look ahead one symbol
SLR= use of LR(0)−items with 1−symbol lookahead

- SLR table construction:

  Expand the grammar with $S' \rightarrow S\$$

  find LR(0) items

  find closure of items

  find set of items from closures

  construct the DFA from sets of items

  find FIRST and FOLLOW sets

  construct ACTION and GOTO tables

- LR(0) ITEM: progress of passing over the RHS of a rule

LR(0)-item($A \rightarrow uv$) $= \{A \rightarrow \cdot uv, A \rightarrow u \cdot v, A \rightarrow uv\cdot\}$

LR(0)-item($A \rightarrow \epsilon$) $= \{A \rightarrow \cdot\}$

eg. for the grammar G_{AE}:

$S{\rightarrow}\cdot A$		$S{\rightarrow}A\cdot$
$A{\rightarrow}\cdot T$		$A{\rightarrow}T\cdot$
$A{\rightarrow}\cdot A+T$		$A{\rightarrow}A\cdot+T$
$A{\rightarrow}A+\cdot T$		$A{\rightarrow}A+T\cdot$
$T{\rightarrow}\cdot b$	$T{\rightarrow}b\cdot$	$T{\rightarrow}\cdot(A)$ $\ldots$

- CLOSURE OF an LR(0) ITEM: All the states of parsing that can be reached from a state.

closure$(S\rightarrow\cdot A) = \{S\rightarrow\cdot A, A\rightarrow\cdot T, A\rightarrow\cdot A + T, T\rightarrow\cdot b, T\rightarrow\cdot(A)\}$

- goto(item,symbol)= the set of states one can go from the item by consuming the symbol (note: this is not the GOTO table!)

goto$(A\rightarrow A\cdot + T, +) = \{A\rightarrow A + \cdot T, T\rightarrow\cdot b, T\rightarrow\cdot(A)\}$

- SETS OF ITEMS= possible states of parsing. Since there are finitely many RHS, there will be finitely many combinations.

initially: $S' \rightarrow \cdot S\$$.

Find out where to go from goto(item,X) for all $X \in V \cup \Sigma$. Add this to sets of items.

Each set is a possible state of SLR parsing (GOTO/ACTION states).

- eg.: $I_0 = \{S' \rightarrow \cdot S\$, S \rightarrow \cdot A, A \rightarrow \cdot T, A \rightarrow \cdot A + T, T \rightarrow \cdot b, T \rightarrow \cdot (A)\}$

  $I_1 = goto(I_0, S) = \{S' \rightarrow S \cdot \$\}$

  $I_2 = goto(I_0, A) = \{S \rightarrow A \cdot, A \rightarrow A \cdot + T\}$

  $I_3 = goto(I_0, T) = \{A \rightarrow T \cdot\}$

  $I_4 = goto(I_0, b) = \{T \rightarrow b \cdot\}$

$I_5 = goto(I_0, () = \{T \rightarrow (\cdot A), A \rightarrow \cdot T, A \rightarrow \cdot A + T, T \rightarrow \cdot b, T \rightarrow \cdot (A)\}$

$I_6 = goto(I_1, \$) = \{S' \rightarrow S\$\cdot\}$

$I_7 = goto(I_2, +) = \{A \rightarrow A + \cdot T, T \rightarrow \cdot b, T \rightarrow \cdot (A)\}$

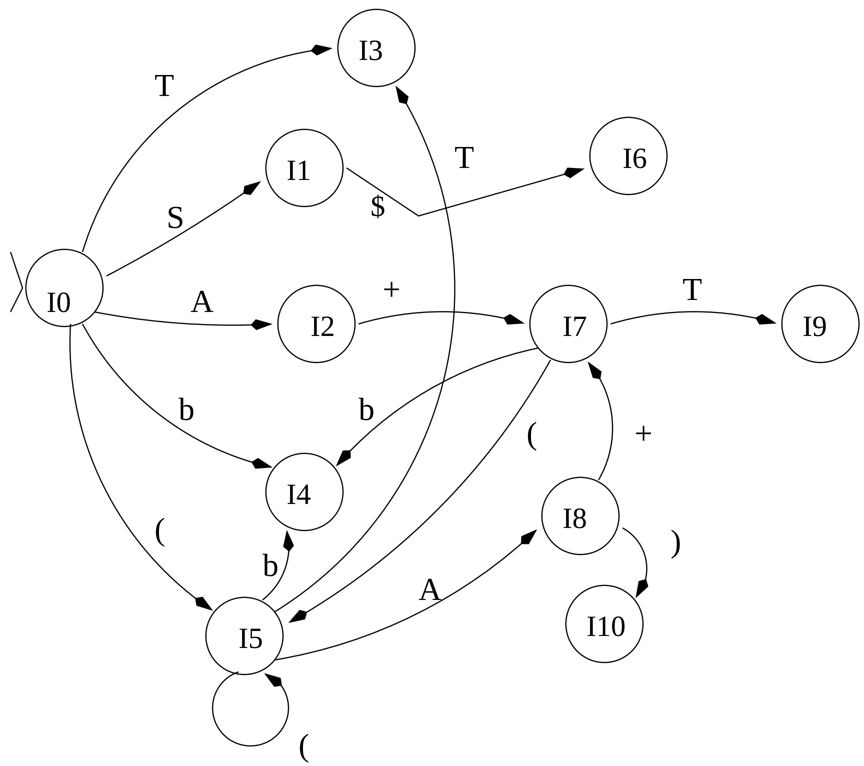
$I_8 = goto(I_5, A) = \{T \rightarrow (A \cdot), A \rightarrow A \cdot + T\}$

$goto(I_5, T) = I_3 \;\; goto(I_5, b) = I_4 \;\; goto(I_5, () = I_5$

$I_9 = goto(I_7, T) = \{A \rightarrow A + T \cdot\}$

$goto(I_7, b) = I_4 \; goto(I_7, () = I_5 \; goto(I_8, +) = I_7$

$I_{10} = goto(I_8,)) = \{T \rightarrow (A) \cdot\}$



The GOTO function as a FA

(GOTO table is a subset which only contains variable transitions)

- FOLLOW sets for G_{AE}

  FOLLOW$(S) = \{\$\}$ FOLLOW$(A) = \{), +, \$\}$

  FOLLOW$(T) = \{\$, +,)\}$

- setting up the ACTION table

for each I_i, if $A \rightarrow \alpha \cdot a \beta$ is in I_i, ACTION[i,a]=shift j (where goto(I_i,a)=I_j)

for each $A \rightarrow \alpha \cdot$, ACTION[$I_i$,a]=reduce by $A \rightarrow \alpha$ for all $a \in FOLLOW(A)$

	ACTION					GOTO		
	b	(	)	+	$	S	A	T
0	sh4	sh5				1	2	3
1					accept			
2				sh7	rS->A			
3			rA->T	rA->T	rA->T			
4			rT->b	rT->b	rT->b			

5	sh4	sh5				8	3
6					accept		
7	sh4	sh5					9
8			sh10	sh7			
9			rA->A+T	rA->A+T	rA->A+T		
10			rT->(A)	rT->(A)	rT->(A)		

parsing (b+b) with SLR table

stack	input	action
0	(b+b)$	sh5
0(5	b+b)$	sh4
0(5b4	+b)$	rT->b
0(5T3		rA->T
0(5A8		sh7
0(5A8+7	b)$	sh4
0(5A8+7b4	)$	rT->b
0(5A8+7T9		rA->A+T
0(5A8		sh10
0(5A8)10	$	rT->(A)
0T3		rA->T
0A2		rS->A
0S1		accept

- A grammar with no conflict in the SLR tables is SLR(1).

- Reductions and new top determine the next viable prefix.
- In general, direct DFA construction is cumbersome. Design a NFA with empty transitions and convert to DFA.
- General LR(k) parsing is an extension of SLR where LR(k)-items are used.