Compiler Construction

Lecture 6: SLR(1) and LR(1)

Jeremy Yallop

jeremy.yallop@cl.cam.ac.uk

Lent 2026

Bottom-up parsing components

Configurations $\$\alpha, w\$$ with **actions**:

$$\$\alpha, xw\$ \xrightarrow{\text{shift } x} \$\alpha x, w\$$$

$$\$\alpha\beta, w\$ \xrightarrow{\text{reduce } A \to \beta} \$\alpha A, w\$$$

Items $A \to \beta \bullet \gamma$ with **transitions**:

$$\boxed{A \to \beta \bullet c\gamma} \xrightarrow{c} \boxed{A \to \beta c \bullet \gamma}$$

$$\boxed{A \to \beta \bullet B\gamma} \xrightarrow{B} \boxed{A \to \beta B \bullet \gamma}$$

$$\boxed{A \to \beta \bullet B\gamma} \xrightarrow{\epsilon} \boxed{B \to \bullet \alpha_i}$$

A **parsing algorithm**:

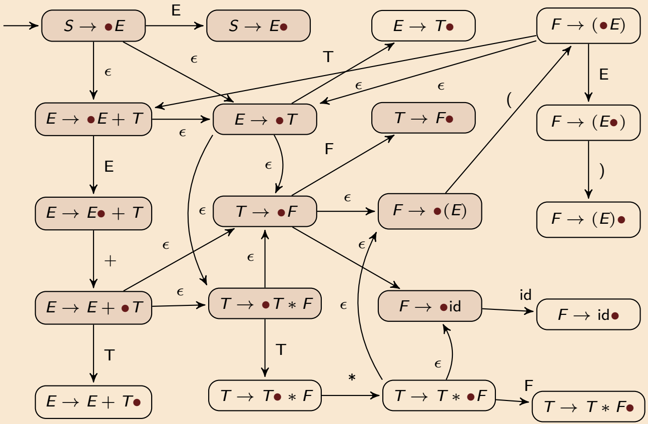
$c :=$ NextToken()

while true:

- $\alpha :=$ the stack
- if $A \to \beta \bullet c\gamma \in \delta_G(q_0, \alpha)$
  then SHIFT c; $c :=$ NextToken()
- if $A \to \beta \bullet \in \delta_G(q_0, \alpha)$
  then REDUCE via $A \to \beta$
- if $S \to \beta \bullet \in \delta_G(q_0, \alpha)$
  then ACCEPT (if no more input)
- if none of the above
  then ERROR

non-deterministic

Using the NFA



stack	input
$\$$	$(x+y)\$$
$\$($	$x+y)\$$
$\$(x$	$+y)\$$
$\$(F$	$+y)\$$
$\$(T$	$+y)\$$
$\$(E$	$+y)\$$
$\$(E+$	$y)\$$
$\$(E+y$	$)\$$
$\$(E+F$	$)\$$
$\$(E+T$	$)\$$
$\$(E$	$)\$$
$\$(E)$	$\$$
$\$F$	$\$$
$\$T$	$\$$
$\$E$	$\$$
$\$S$	$\$$

Example: $\delta_G(S \to \bullet E, E+F) = \{T \to F\bullet\}$, so reduce with $T \to F$.

Example: $\delta_G(S \to \bullet E, (E) = \{F \to (E\bullet)\}$, so shift).

Making the algorithm deterministic

Two sources of **nondeterminism**:

1. The **NFA**

$$\begin{tikzcd} \boxed{F \to \bullet(E)} \arrow[d, "("] & \boxed{T \to \bullet F} \arrow[l, "\epsilon"'] \arrow[d, "\epsilon"] \\ \boxed{F \to (\bullet E)} & \boxed{F \to \bullet id} \end{tikzcd}$$

Solution: convert to a **DFA**

$$\begin{tikzcd} \boxed{\begin{array}{l} T \to \bullet F \\ F \to \bullet(E) \\ F \to \bullet id \end{array}} \arrow[d, "("] \\ \boxed{F \to (\bullet E)} \end{tikzcd}$$

2. **Conflicts**

$$\$\alpha E + T, x\$ \xrightarrow{\text{reduce } E \to E+T}$$
$$\$\alpha E + T, x\$ \xrightarrow{\text{reduce } E \to T}$$
$$\$\alpha E + T, x\$ \xrightarrow{\text{shift } x}$$

Solution: make a **deterministic choice**

using the **input** (lookahead)	using the **grammar** (FIRST and FOLLOW)

Two approaches: SLR(1) and LR(1)

The DFA

The easy part: NFA $\rightarrow$ DFA

Grammar G_2'

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid \text{id}
\end{aligned}
$$

$$
\text{DFA start state} = \epsilon\text{–closure}(\{S \rightarrow \bullet E\}) = \boxed{\begin{array}{l} S \rightarrow \bullet E \\ E \rightarrow \bullet E + T \\ E \rightarrow \bullet T \\ T \rightarrow \bullet T * F \\ T \rightarrow \bullet F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet \text{id} \end{array}}
$$

Transition function:

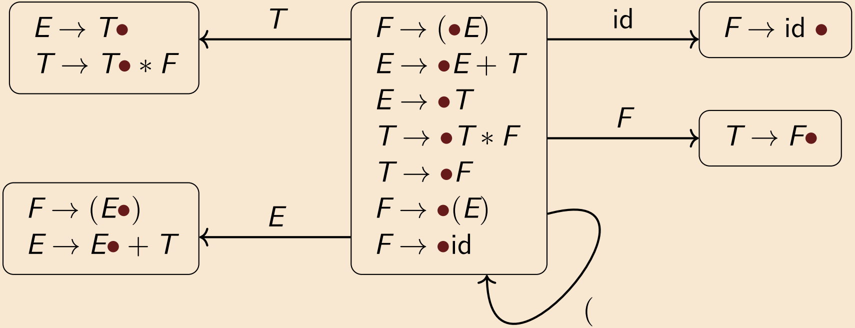
$$
\delta(I, X) = \epsilon\text{–closure}(\{A \rightarrow \alpha X \bullet \beta \mid A \rightarrow \alpha \bullet X \beta \in I\})
$$

*(**NB**: this is just the powerset/subset construction for converting NFAs to DFAs.)*

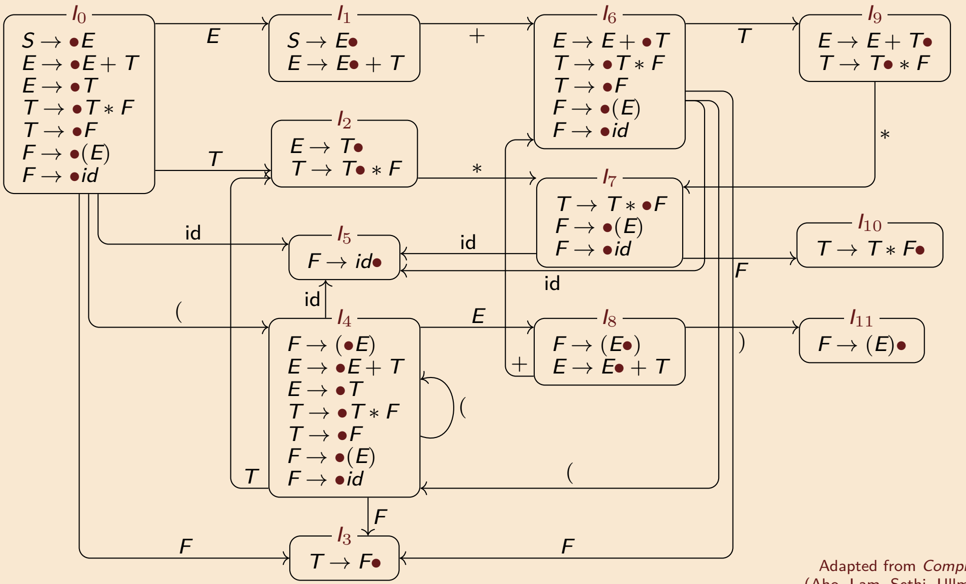
Some DFA transitions for grammar G_2

Grammar G_2'

$S \to E$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid \mathrm{id}$



Full DFA for the stack language of G_2



Adapted from *Compilers* (Aho, Lam, Sethi, Ullman)

SLR(1)

The state I_2 has a **shift/reduce conflict**.

I_2

$E \to T\bullet$
$T \to T\bullet * F$

The **S**imple **LR(1)** approach resolves the conflict using the next token c:

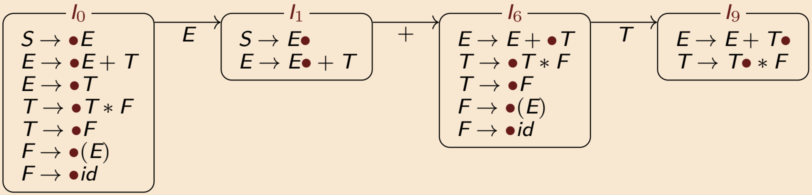
shift if $c = *$

reduce with $E \to T$ only if $c \in \text{FOLLOW}(E) = \{(, +, \$\}$.

Deterministic SLR(1) parsing

Recall: when the stack contains α, the parser is in state $\delta(I_0, \alpha)$. For example,

$$\delta(I_0, E + T) \quad = \quad I_9$$



Let I be the current state and c the next token. Then:

1. When $A \to \beta \bullet c \gamma \in I$ then **shift** c onto stack
2. When $A \to \beta \bullet \in I$ and $c \in \text{FOLLOW}(A)$ then **reduce** with $A \to \beta$

Replay parsing of $(x + y)$ using SLR(1) actions

stack,	input	state	action	reason
$\$$	$(x + y)\$$	I_0	shift (	
$\$($	$x + y)\$$	I_4	shift x	
$\$(x$	$+y)\$$	I_5	reduce $F \to \text{id}$	
$\$(F$	$+y)\$$	I_3	reduce $T \to F$	
$\$(T$	$+y)\$$	I_2	reduce $E \to T$	
$\$(E$	$+y)\$$	I_8	shift $+$	
$\$(E+$	$y)\$$	I_6	shift y	
$\$(E + y$	$)\$$	I_5	reduce $F \to \text{id}$	
$\$(E + F$	$)\$$	I_3	reduce $T \to F$	
$\$(E + T$	$)\$$	I_9	reduce $E \to E$ ·	
$\$(E$	$)\$$	I_8	shift)	
$\$(E)$	$\$$	I_{11}	reduce $F \to (E$	
$\$F$	$\$$	I_3	reduce $T \to F$	
$\$T$	$\$$	I_2	reduce $F \to E$	
$\$E$	$\$$	I_1	reduce $S \to E$	

Parsing with states

Idea: don't restart the DFA at every step

Previous approach

Maintain a **stack of symbols**

$$\$\ (\ E\ +\ \mathsf{id}$$

At each step:

Use the **full stack**
to find items

New approach

Maintain a **stack of states**

$$0\ 4\ 1\ 6\ 5$$

At each step:

Use the **top of the stack**
to find actions

LR parsing with DFA states on the stack

tok := NextToken()

while true:

state := TopStackState()

if $\textsc{action}[\text{state}, \text{tok}] = \textsc{shift}$ state

then push state

tok := NextToken()

else if $\textsc{action}[\text{state}, \text{tok}] = \textsc{reduce}\ A \to \beta$

then pop $|\beta|$ states

push $\textsc{goto}[\text{TopStackState}(), A]$

else if $\textsc{action}[\text{state}, \text{tok}] = \textsc{accept}$

then accept and exit

else $\textsc{error}$

Constructing ACTION and GOTO for SLR(1)

If $A \to \alpha \bullet a\beta \in I_i$ and $\delta(I_i, a) = I_j$

then $\textsc{action}[i, a] = \textsc{shift}\ j$.

If $A \to \alpha \bullet \in I_i$ and $A \neq S$

then for each $a \in \textsc{follow}(A)$,

$\textsc{action}[i, a] = \textsc{reduce}\ A \to \alpha$

If $S \to \alpha \bullet \in I_i$

then $\textsc{action}[i, \$] = \textsc{accept}$

If $\delta(I_i, A) = I_j$

then $\textsc{goto}[i, A] = j$

(ACTION resolves conflicts; GOTO records nonterminal transitions $I_i \xrightarrow{A} I_j$)

ACTION and GOTO for G_2'

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		$r5$	$r5$		$r5$	$r5$			

Adapted from *Compilers*
(Aho, Lam, Sethi, Ullman)

Example parse

STACK	SYMBOLS	INPUT	ACTION	
0		id $*$ id $+$ id\$	shift	
0 5	id	$*$ id $+$ id\$	reduce $F \to$ id	$\text{GOTO}[0, F] = 3$
0 3	F	$*$ id $+$ id\$	reduce $T \to F$	$\text{GOTO}[0, T] = 2$
0 2	T	$*$ id $+$ id\$	shift	
0 2 7	$T *$	id $+$ id\$	shift	
0 2 7 5	$T *$ id	$+$ id\$	reduce $F \to$ id	$\text{GOTO}[7, F] = 10$
0 2 7 10	$T * F$	$+$ id\$	reduce $T \to T * F$	$\text{GOTO}[0, T] = 2$
0 2	T	$+$ id\$	reduce $E \to T$	$\text{GOTO}[0, E] = 1$
0 1	E	$+$ id\$	shift	
0 1 6	$E +$	id\$	shift	
0 1 6 5	$E +$ id	\$	reduce $F \to$ id	$\text{GOTO}[6, F] = 3$
0 1 6 3	$E + F$	\$	reduce $T \to F$	$\text{GOTO}[6, T] = 9$
0 1 6 9	$E + T$	\$	reduce $E \to E + T$	$\text{GOTO}[0, E] = 1$
0 1	E	\$	accept	

Adapted from *Compilers*
(Aho, Lam, Sethi, Ullman)

The limits of SLR(1)

Recap & plan
DFA
SLR(1)
Parsing with states
SLR(1) limits
LR(1)

A new example grammar (for *assignment expressions*):

$$G_4 = \langle N_4, T_4, P_4, S' \rangle$$

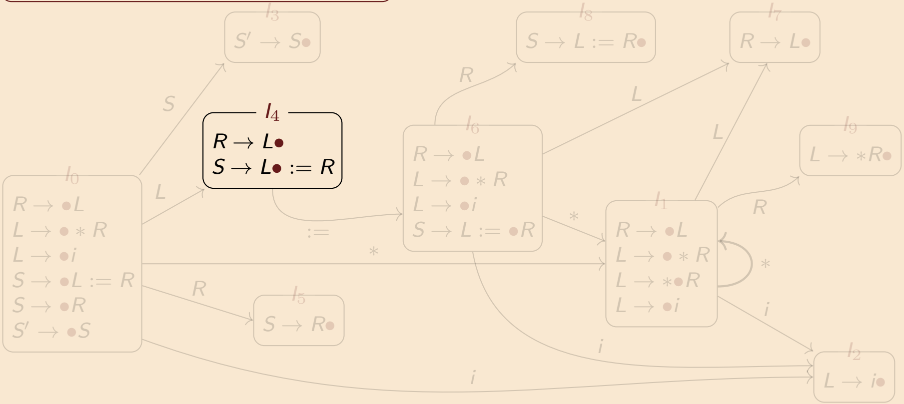
$$N_4 = \{S', S, L, R\}$$

$$T_4 = \{*, :=, id\}$$

$$\begin{aligned} P_4: \quad & S' \to S\,\$ \\ & S \to L := R \mid R \\ & L \to *R \mid \mathrm{id} \\ & R \to L \end{aligned}$$

LR(0) DFA for grammar G_4

Ambiguity
In state 4 there is a shift/reduce conflict between $S \to L\bullet := R$ and $R \to L\bullet$



SLR(1) cannot resolve this conflict.

Suppose we see := in the input in state I_4. Then:

shifting is valid

Since $S \to L\bullet := R \in I_4$,
and $\delta(I_4, \text{":="}) = I_6$,

$\quad \text{ACTION}[4, \text{":="}] = \text{shift } 6$

reducing is valid

Since $R \to L\bullet \in I_4$
and $\text{":="} \in \text{FOLLOW}(R)$,

$\quad \text{ACTION}[4, \text{":="}] = \text{reduce } R \to L$

LR(1)

Beyond SLR(1): LR(1)

With SLR(1) there may be

shift-reduce conflicts	**reduce-reduce** conflicts

when ACTION and GOTO are not uniquely defined.

Options: fix the grammar, or use a more powerful parsing technique.

LR(1) parsing extends LR(0) items with an explicit lookahead token a:

$$\overbrace{\underbrace{A \rightarrow \alpha \bullet \beta}_{\text{LR(0) item}}, \underbrace{a}_{\text{lookahead token}}}^{\text{LR(1) item}}$$

Define an NFA with LR (1) items as states

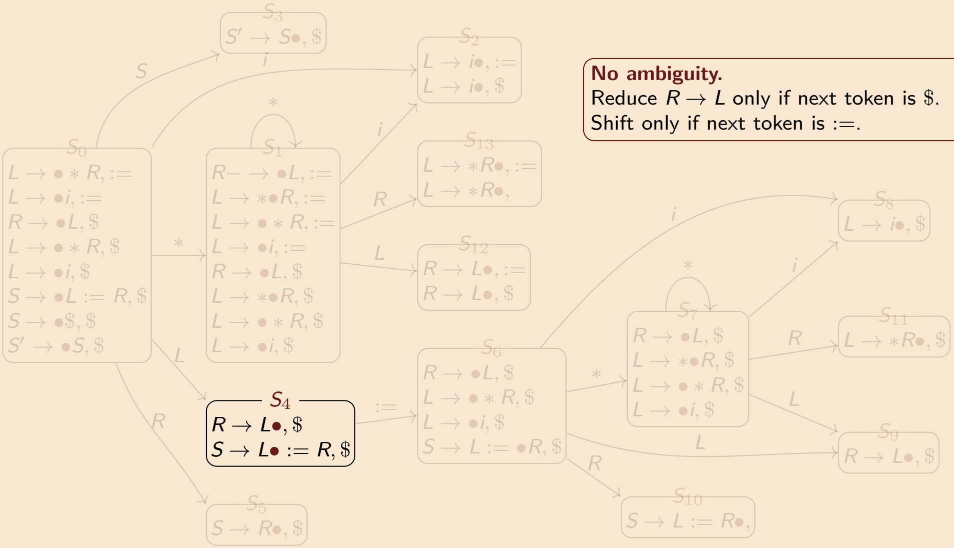
$$\boxed{A \to \alpha \bullet c\beta,\ a} \xrightarrow{\quad c \quad} \boxed{A \to \alpha c \bullet \beta,\ a}$$

$$\boxed{A \to \alpha \bullet B\beta,\ a} \xrightarrow{\quad B \quad} \boxed{A \to \alpha B \bullet \beta,\ a}$$

For each $b \in \textsc{first}(\beta a)$:

$$\boxed{A \to \alpha \bullet B\beta,\ a} \xrightarrow{\quad \epsilon \quad} \boxed{B \to \bullet \gamma,\ b}$$

LR(1) DFA for grammar G_4



No ambiguity.
Reduce $R \to L$ only if next token is $\$$.
Shift only if next token is $:=$.

Constructing ACTION and GOTO for LR(1)

If $[A \to \alpha \bullet a\beta, b] \in I_i$ and $\delta(I_i, a) = I_j$

then $\text{ACTION}[i, a] = \text{SHIFT} \;\; j$.

If $[A \to \alpha \bullet, b] \in I_i$ and $A \neq S$,

then $\text{ACTION}[i, b] = \text{REDUCE} \;\; A \to \alpha$

If $[S \to \alpha \bullet, \$] \in I_i$

then $\text{ACTION}[i, \$] = \text{ACCEPT}$

If $\delta(I_i, A) = I_j$

then $\text{GOTO}[i, A] = j$.

Key change from SLR(1): use **lookahead** (i.e. FIRST), not FOLLOW, to select REDUCE actions

SLR(1) vs LR(1)

SLR(1)

If $[A \to \alpha\bullet] \in I_i$ and $A \neq S$
then for each $a \in \textsc{follow}(A)$,
$\textsc{action}[i, a] = \text{reduce } A \to \alpha$

LR(1)

If $[A \to \alpha\bullet, b] \in I_i$ and $A \neq S$
then
$\textsc{action}[i, b] = \text{reduce } A \to \alpha$

NB: b used *only* for reductions, not for shifts

LR(1) more powerful than SLR(1)

LR(1) DFA may have a very large number of states.
LALR[1] optimises DFA, collapsing states (but can give strange error messages)

[1]Implemented in yacc; not covered here

Next time: translation