Bottom-up parsing

Reconstruct, in reverse order, the rightmost derivation of a word from the yield of the tree to its root

Various techniques available (e.g., LR(1), LALR(1), SLR(1))

Bottom-up parsing

All bottom-up parsing techniques share fundamental elements:

- Always $\mathcal{P}$ extended to $\mathcal{P}'$ by adding the production $S' \rightarrow S$ where S' is a new non-terminal
- Same **shift/reduce algorithm** for parsing
- Always **characteristic automata** as controllers of the parsing algorithm

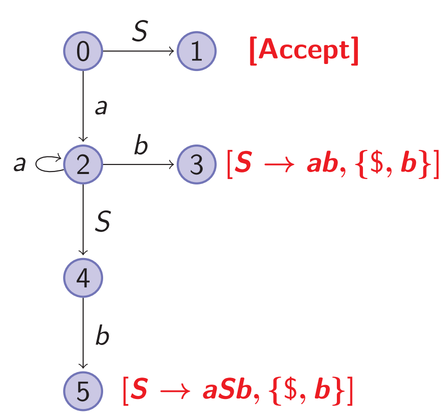
Bottom-up parsing

Depending on the technique, characteristic automata encode finer or coarser information

The finer the information, the bigger characteristic automata, the more powerful the parsing techinque, the bigger the set of analyzed grammars

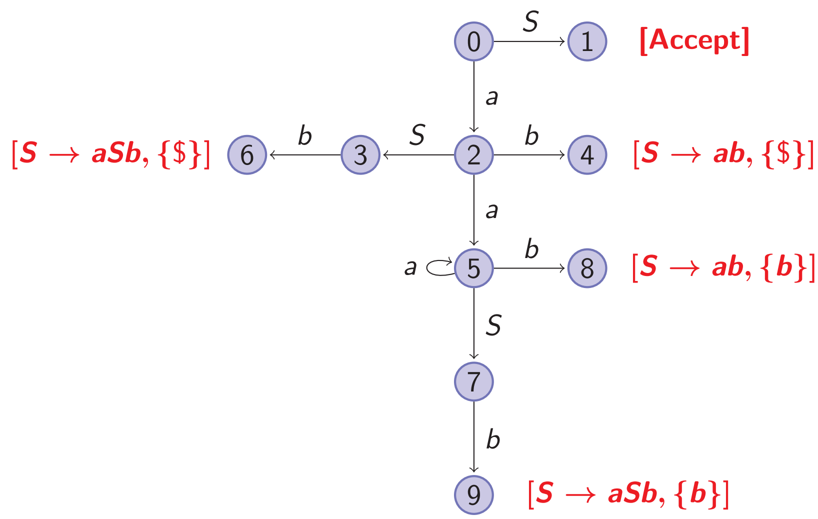
Example

LR(0)-AUTOMATON FOR $S \to aSb \mid ab$



Example

LR(1)-AUTOMATON FOR $S \to aSb \mid ab$



LR(0)-automata

States are sets of **items**

Items:

- LR(0)-items: $A \to \alpha \cdot \beta$

Characteristic automata:

- LR(0)-automata: states are sets of LR(0)-items

LR(0)-items

Consider the LR(0)-item $S' \to \cdot S$

It intuitively means that we have not yet seen any symbol of the word that we are going to parse, and parsing will be successful only if the word we analyze derives from S

Then the item $S' \to \cdot S$ is surely in the initial state of the characteristic automaton, say P_0

Other items should be in P_0 too!

LR(0)-items

Consider the grammar $S \to aSb \mid ab$

If we start analyzing a word and expect to see a derivation from S, then we expect to see either a derivation from aSb or a derivation from ab

Then the LR(0)-items

- $S \to \cdot aSb$
- $S \to \cdot ab$

Should be in P_0 as well

Closure of sets of LR(0)-items

Let P be a set of LR(0)-items

$\mathrm{closure}_0(P)$ is the smallest set of items that satisfies the following equation

$\mathrm{closure}_0(P) = P \cup$
$\{B \to \cdot\gamma$ such that $A \to \alpha \cdot B\beta \in \mathrm{closure}_0(P)$ and $B \to \gamma \in \mathcal{P}'\}$

Fixed point computation: initialize $\mathrm{closure}_0(P)$ as P, then add items as needed, up to reaching the fixed point

$\mathrm{closure}_0(P)$
EXAMPLE

Take

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

Compute $\mathrm{closure}_0(\{E' \to \cdot E\})$

$\mathrm{closure}_0(P)$

EXAMPLE

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

$\mathrm{closure}_0(P) = P \cup$
$\{B \rightarrow \cdot\gamma$ such that $A \rightarrow \alpha \cdot B\beta \in \mathrm{closure}_0(P)$ and $B \rightarrow \gamma \in \mathcal{P}'\}$

- Init: $\mathrm{closure}_0(\{E' \rightarrow \cdot E\}) = \{E' \rightarrow \cdot E\}$
- Add $E \rightarrow \cdot E + T$ and $E \rightarrow \cdot T$
- Add $T \rightarrow \cdot T * F$ and $T \rightarrow \cdot F$
- Add $F \rightarrow \cdot(E)$ and $F \rightarrow \cdot id$

$\mathrm{closure}_0(P)$

function $\mathrm{closure}_0(P)$
  tag every item in P as unmarked ;
  while *there is an unmarked item I in P* **do**
    mark I ;
    if I *has the form* $A \rightarrow \alpha \cdot B\beta$ **then**
      foreach $B \rightarrow \gamma \in \mathcal{P}'$ **do**
        if $B \rightarrow \cdot\gamma \notin P$ **then**
          add $B \rightarrow \cdot\gamma$ as an unmarked item to P ;
  return P ;

Construction of LR(0)-automaton

Construct the automaton by populating a collection of states while definining the transition function

$P_0 = \mathrm{closure}_0(\{S' \to \cdot S\})$

If an already collected state P contains an item of the form $A \to \alpha \cdot Y\beta$

Then there is a transition from P to a state Q which contains the item $A \to \alpha Y \cdot \beta$

And, since Q contains $A \to \alpha Y \cdot \beta$, it also contains all the items in $\mathrm{closure}_0(\{A \to \alpha Y \cdot \beta\})$

Example

$$\begin{array}{lcl} S & \to & aABe \\ A & \to & Abc \mid b \\ B & \to & d \end{array}$$

State 0:

$\boxed{S' \to \cdot S}$

$\boxed{S \to \cdot aABe}$

The items in state 0 have the marker at the left of S and of a

Then, if not there yet, we must add to the collection two more states:

- $\tau(0, S)$
- $\tau(0, a)$

Example

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$

0

$\boxed{S' \rightarrow \cdot S}$

$\boxed{S \rightarrow \cdot aABe}$

$\tau(0, S) = 1$

$\boxed{S' \rightarrow S \cdot}$

No transition from this one

$\tau(0, a) = 2$

$\boxed{S \rightarrow a \cdot ABe}$

$\boxed{\begin{array}{l} A \rightarrow \cdot Abc \\ A \rightarrow \cdot b \end{array}}$

If not there yet, add $\tau(2, A)$ and $\tau(2, b)$

Example

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$

2

$\boxed{S \rightarrow a \cdot ABe}$

$\boxed{\begin{array}{l} A \rightarrow \cdot Abc \\ A \rightarrow \cdot b \end{array}}$

$\tau(2, A) = 3$

$\boxed{\begin{array}{l} S \rightarrow aA \cdot Be \\ A \rightarrow A \cdot bc \end{array}}$

$\boxed{B \rightarrow \cdot d}$

If not there yet, add $\tau(3, B)$, $\tau(3, b)$ and $\tau(3, d)$

$\tau(2, b) = 4$

$\boxed{A \rightarrow b \cdot}$

No transition from this one

Example

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$

3

$$\boxed{\begin{array}{l} S \rightarrow aA \cdot Be \\ A \rightarrow A \cdot bc \end{array}}$$
$$\boxed{B \rightarrow \cdot d}$$

$\tau(3, B) = 5$

$\boxed{S \rightarrow aAB \cdot e}$ If not there yet, add $\tau(5, e)$

$\tau(3, b) = 6$

$\boxed{A \rightarrow Ab \cdot c}$ If not there yet, add $\tau(6, c)$

$\tau(3, d) = 7$

$\boxed{B \rightarrow d\, \cdot}$ No transition from this one

Example

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$

5 $\boxed{S \rightarrow aAB \cdot e}$ 6 $\boxed{A \rightarrow Ab \cdot c}$

$\tau(5, e) = 8$

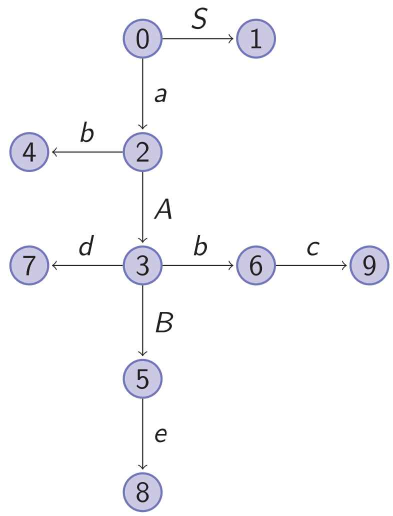
$\boxed{S \rightarrow aABe\, \cdot}$ No transition from this one

$\tau(6, c) = 9$

$\boxed{A \rightarrow Abc\, \cdot}$ No transition from this one either

Example

$$\begin{array}{lcl} S & \rightarrow & aABe \\ A & \rightarrow & Abc \mid b \\ B & \rightarrow & d \end{array}$$



Construction of LR(0)-automata

initialize the collection $\mathcal{Q}$ to contain $P_0 = \mathrm{closure}_0(\{S' \rightarrow \cdot S\})$;
tag P_0 as unmarked;
while *there is an unmarked state P in $\mathcal{Q}$* **do**
    mark P ;
    foreach *Y on the right side of the marker in some item of P* **do**
        Compute in *Tmp* the kernel-set of the Y-target of P;
        if *$\mathcal{Q}$ already contains a state Q whose kernel is Tmp* **then**
            Let Q be the Y-target of P;
        else
            Add $\mathrm{closure}_0(\mathit{Tmp})$ to $\mathcal{Q}$ as an unmarked state;
            Let $\mathrm{closure}_0(\mathit{Tmp})$ be the Y-target of P;

LR(1)-automata

Richer than LR(0)-automata

States are sets of **LR(1)-items**

LR(1)-items: $[A \rightarrow \alpha \cdot \beta, \Delta]$ where $L \subseteq T \cup \{\$\}$

Shift/reduce parsing tables

The moves of the shift/reduce algorithm are driven by a parsing table

The parsing table has one row for each state of the chosen characteristic automaton, and one colum for each symbol in $V \cup \{\$\}$

- Shift moves depend on the transition function of the automanton
- Reduction of the production $A \rightarrow \beta$ is done when the parser is in a state containing the **reducing item** for $A \rightarrow \beta$
  - Item $A \rightarrow \beta\cdot$ in the case of LR(0)-automata
  - Item $[A \rightarrow \beta\cdot, \Delta]$ in the case of LR(1)-automata

  Reductions depend on the **lookahead function** $\mathcal{LA}$ which returns a subset of $V \cup \{\$\}$ for each pair $(P, A \rightarrow \beta)$ such that P contains the reducing item for $A \rightarrow \beta$

Shift/reduce parsing tables

Distinct choices of characteristic automaton and of lookahead function drive the construction of parsing tables for **distinct techniques** of bottom-up parsing (e.g., SLR(1), LR(1), LALR(1))

The class of analyzable grammars depends on the above choice

The size of parsing tables depends on the above choice

The procedure to fill in the table is always **the same**

The parsing algorithm is always **the same**

Shift/reduce parsing tables

CONSTRUCTION

Fill in each entry (P, Y) after the following rules

- Insert *"Shift Q"* if Y is a terminal and $\tau(P, Y) = Q$
- Insert *"Reduce $A \to \beta$"* if P contains the reducing item for $A \to \beta$ and $Y \in \mathcal{LA}(P, A \to \beta)$
- Set to *"Accept"* if P contains the **accepting item** and $Y = \$$
  - Item $S' \to S\cdot$ in the case of LR(0)-automata
  - Item $[S' \to S\cdot, \Delta]$ in the case of LR(1)-automata
- Set to *"Goto Q"* if Y is a nonterminal and $\tau(P, Y) = Q$

Shift/reduce parsing tables

CONFLICTS

The table can have entries multiply-defined

- **s/r conflict** (shift/reduce conflict): If the entry contains both a shift and a reduce move
- **r/r conflict** (reduce/reduce conflict): If the entry contains two reduce moves for distinct productions

SLR(1) parsing tables

SLR(1) parsing tables are obtained by taking:

- Characteristic automaton: LR(0)-automaton
- Lookahead function: $\mathcal{LA}(P, A \to \beta) = \mathrm{follow}(A)$ for every $A \to \beta\cdot \in P$

$\mathcal{G}$ **is SLR(1)** iff its SLR(1) parsing table has no conflict

Example

Construct the SLR(1) parsing table for

$$
\begin{array}{lcl}
S & \to & aABe \\
A & \to & Abc \mid b \\
B & \to & d
\end{array}
$$

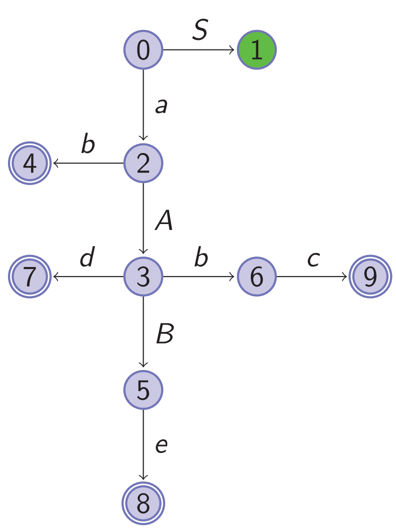
Example

LR(0)-automaton for

$$
\begin{array}{lcl}
S & \to & aABe \\
A & \to & Abc \mid b \\
B & \to & d
\end{array}
$$

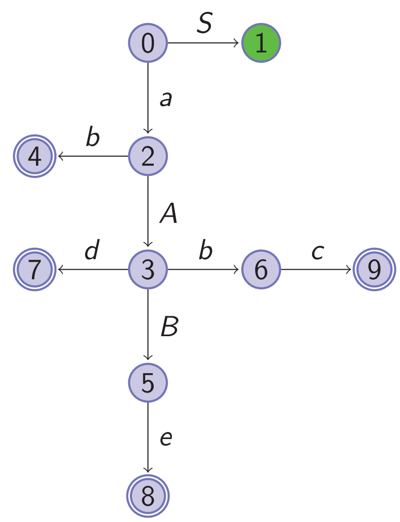
Where

- States containing reducing items are drawn as final states
- States containing the accepting item are colored in green



Example

- $4 = \{A \to b\cdot\}$
- $7 = \{B \to d\cdot\}$
- $8 = \{S \to aABe\cdot\}$
- $9 = \{A \to Abc\cdot\}$



Example

1. $S \to aABe$
2. $A \to Abc$
3. $A \to b$
4. $B \to d$

	a	b	c	d	e	\$	S	A	B
0	s2						1		
1						Acc			
2		s4						3	
3		s6		s7					5
4		r3		r3					
5					s8				
6			s9						
7					r4				
8						r1			
9		r2		r2					

The grammar is SLR(1)

Shift/reduce parsing
ALGORITHM

- **Input:**
  string w; shift/reduce parsing table M for $\mathcal{G} = (V, T, S, \mathcal{P})$
- **Output:**
  rightmost derivation of w in reverse order if $w \in \mathcal{L}(\mathcal{G})$, error() otherwise
- **Initialization:**
  P_0 onto the state-stack *stSt*; nothing onto the symbol-stack *symSt*; $w\$$ in the input buffer

FORMAL LANGUAGES AND COMPILERS Paola Quaglia, 2022

Shift/reduce parsing
ALGORITHM

```
let b be the first symbol in the input buffer;
while true do
    let S be the top of stSt;
    if M[S, b] = "Shift T" then
        push b onto symSt;
        push T onto stSt;
        let b be the next symbol in the input buffer;
    else if M[S, b] = "Reduce A → β" then
        pop |β| symbols off symSt; push A onto symSt;
        pop |β| symbols off stSt;
        let temp be the top of stSt;
        push T onto stSt, where T is such that M[temp, A] = "Goto T";
        output "A → β";
    else if M[S, b] = "Accept" then return;
    else error();
```

Example

Parse $w = abbcde$

1. $S \rightarrow aABe$
2. $A \rightarrow Abc$
3. $A \rightarrow b$
4. $B \rightarrow d$

	a	b	c	d	e	\$	S	A	B
0	s2						1		
1						Acc			
2		s4						3	
3		s6		s7					5
4		r3		r3					
5					s8				
6			s9						
7					r4				
8						r1			
9		r2		r2					

Example

$abbcde\$$	$stSt$	0						
	$symSt$							
$\underline{a}bbcde\$$	$stSt$	0	2					
	$symSt$	a						
$\underline{ab}bcde\$$	$stSt$	0	2	4				
	$symSt$	a	b					
$\underline{ab}bcde\$$	$stSt$	0	2	$\cancel{4}$	3			
	$symSt$	a	$\cancel{b}$	A				output $A \rightarrow b$
$\underline{abb}cde\$$	$stSt$	0	2	3	6			
	$symSt$	a	A	b				
$\underline{abbc}de\$$	$stSt$	0	2	3	6	9		
	$symSt$	a	A	b	c			

Example

$\underline{abbc}de\$$	*stSt*	0	2	$\cancel{3}$	$\cancel{6}$	$\cancel{9}$	3	
	symSt	a	$\cancel{A}$	$\cancel{b}$	$\cancel{c}$	A		output $A \to Abc$
$\underline{abbcd}e\$$	*stSt*	0	2	3	7			
	symSt	a	A	d				
$\underline{abbcd}e\$$	*stSt*	0	2	3	$\cancel{7}$	5		
	symSt	a	A	$\cancel{d}$	B			output $B \to d$
$\underline{abbcde}\$$	*stSt*	0	2	3	5	8		
	symSt	a	A	B	e			
$\underline{abbcde}\$$	*stSt*	0	$\cancel{2}$	$\cancel{3}$	$\cancel{5}$	$\cancel{8}$	1	
	symSt	$\cancel{a}$	$\cancel{A}$	$\cancel{B}$	$\cancel{e}$	S		output $S \to aABe$
$\underline{abbcde}\$$	*stSt*	0	1					
	symSt	S			Accept			

Example

The output, taken in reverse order, is the sequence of productions to apply for the rightmost derivation of *abbcde*

Output	Derivation
$S \to aABe$	$S \Rightarrow aABe$
$B \to d$	$\Rightarrow aAde$
$A \to Abc$	$\Rightarrow aAbcde$
$A \to b$	$\Rightarrow abbcde$

SLR(1) parsing
CASE STUDY

Construct the SLR(1) parsing table for

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

Formal Languages and Compilers Paola Quaglia, 2022

SLR(1) parsing
CASE STUDY

2

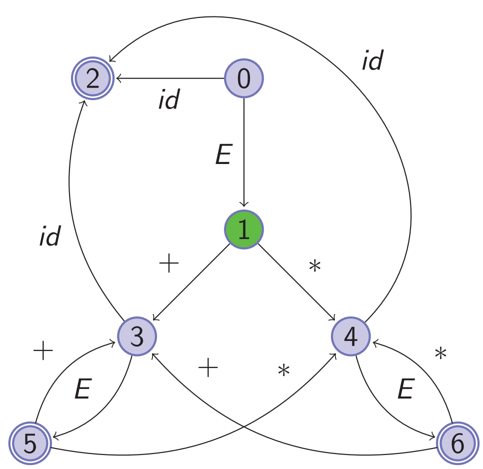
$$\boxed{E \rightarrow id\cdot}$$

5

$$\boxed{\begin{array}{l} E \rightarrow E + E\cdot \\ E \rightarrow E \cdot + E \\ E \rightarrow E \cdot * E \end{array}}$$

6

$$\boxed{\begin{array}{l} E \rightarrow E * E\cdot \\ E \rightarrow E \cdot + E \\ E \rightarrow E \cdot * E \end{array}}$$



SLR(1) parsing
CASE STUDY

1. $E \rightarrow E + E$
2. $E \rightarrow E * E$
3. $E \rightarrow id$

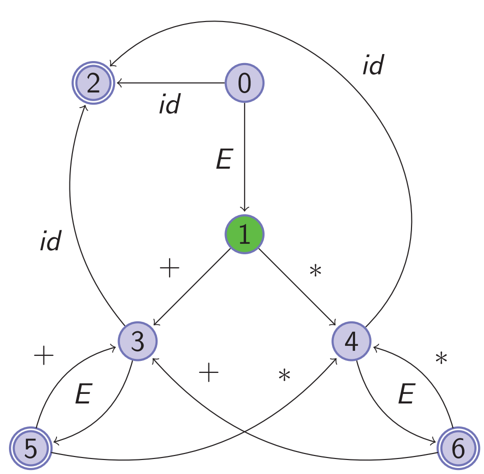
	id	$+$	$*$	\$	E
0	s2				1
1		s3	s4	Acc	
2		r3	r3	r3	
3	s2				5
4	s2				6
5		s3; r1	s4; r1	r1	
6		s3; r2	s4; r2	r2	

Resolving conflicts
CASE STUDY

	$+$	$*$
5	s3; r1	s4; r1
6	s3; r2	s4; r2

If the parser is in state 5 then $E + E$ is on *symSt*

If the parser is in state 6 then $E * E$ is on *symSt*



Resolving conflicts
CASE STUDY

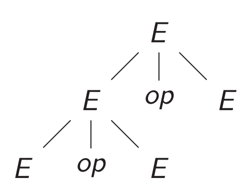
	$+$	$*$
5	s3; r1	s4; r1
6	s3; r2	s4; r2

These conflicts are related to the associativity and precedence of the operators $+$ and $*$

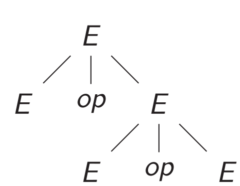
Resolving conflicts
CASE STUDY

Trees encode parentheses: If *op* is left associative

We want



Rather than



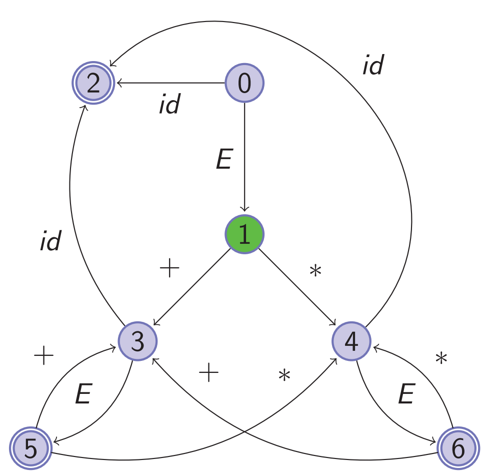
Resolving conflicts
CASE STUDY

Trees encode parentheses: Since $*$ has precedence over $+$ we want

Resolving conflicts
CASE STUDY

The operator $+$ is left associative

	$+$	$*$
5	~~s3~~; r1	s4; r1
6	s3; r2	s4; r2

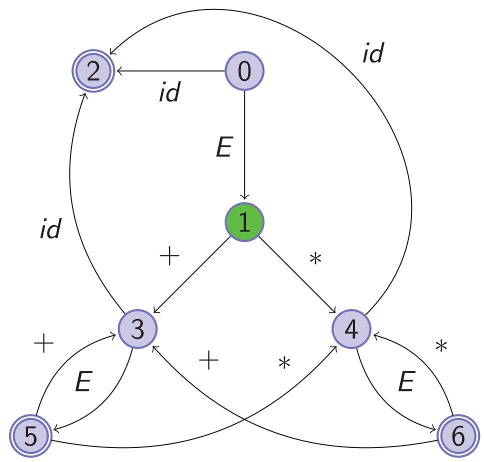


Resolving conflicts

CASE STUDY

The operator $*$ is left associative

	$+$	$*$
5	r1	s4; r1
6	s3; r2	~~s4~~; r2



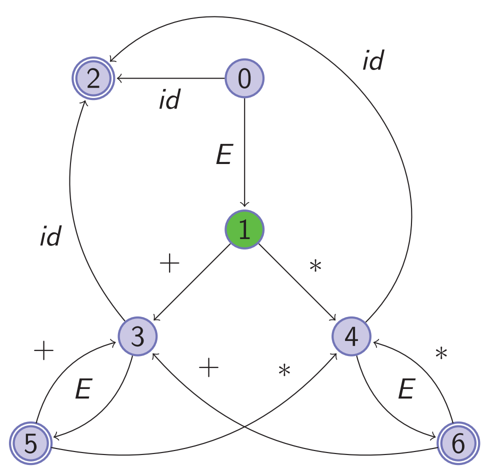
Formal Languages and Compilers Paola Quaglia, 2022

Resolving conflicts

CASE STUDY

The operator $*$ has higher precedence than $+$

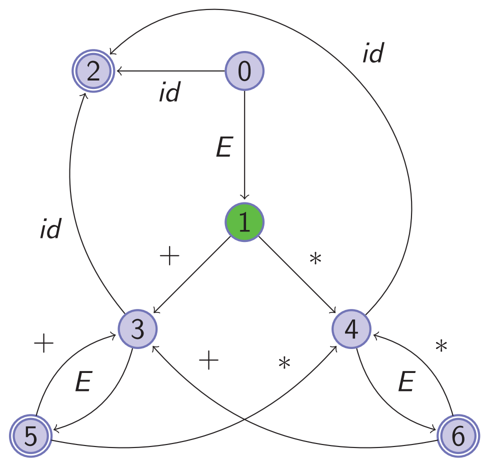
	$+$	$*$
5	r1	s4; ~~r1~~
6	~~s3~~; r2	r2



Resolving conflicts
CASE STUDY

Done!

	$+$	$*$
5	r1	s4
6	r2	r2



Resolving conflicts

Alternative: "just" change the grammar

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

Really?

- Why not $E \rightarrow T + E$ instead?
- Why not $E \rightarrow E * T$ instead?

Time for thought

SLR(1) parsing
ANOTHER CASE STUDY

Construct the SLR(1) parsing table for

$$\begin{array}{lcl} S & \to & aAd \mid bBd \mid aBe \mid bAe \\ A & \to & c \\ B & \to & c \end{array}$$

SLR(1) parsing
ANOTHER CASE STUDY

$$\begin{array}{lcl} S & \to & aAd \mid bBd \mid aBe \mid bAe \\ A & \to & c \\ B & \to & c \end{array}$$

0:

$S' \to \cdot S$
$S \to \cdot aAd$
$S \to \cdot bBd$
$S \to \cdot aBe$
$S \to \cdot bAe$

$1 = \tau(0, S)$:

$S' \to S\,\cdot$

$2 = \tau(0, a)$:

$S \to a \cdot Ad$
$S \to a \cdot Be$
$A \to \cdot c$
$B \to \cdot c$

SLR(1) parsing

ANOTHER CASE STUDY

$$\begin{array}{lcl} S & \to & aAd \mid bBd \mid aBe \mid bAe \\ A & \to & c \\ B & \to & c \end{array}$$

$3 = \tau(0, b)$:
$\boxed{\begin{array}{l} S \to b \cdot Bd \\ S \to b \cdot Ae \end{array}}$
$\boxed{\begin{array}{l} A \to \cdot c \\ B \to \cdot c \end{array}}$

$4 = \tau(2, A)$:
$\boxed{S \to aA \cdot d}$

$5 = \tau(2, B)$:
$\boxed{S \to aB \cdot e}$

$6 = \tau(2, c)$:
$\boxed{\begin{array}{l} A \to c \cdot \\ B \to c \cdot \end{array}}$

$7 = \tau(3, B)$:
$\boxed{S \to bB \cdot d}$

SLR(1) parsing

ANOTHER CASE STUDY

$$\begin{array}{lcl} S & \to & aAd \mid bBd \mid aBe \mid bAe \\ A & \to & c \\ B & \to & c \end{array}$$

$8 = \tau(3, A)$:
$\boxed{S \to bA \cdot e}$

$\tau(3, c) = 6$

$10 = \tau(4, d)$:
$\boxed{S \to aAd \cdot}$

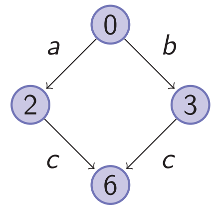
$11 = \tau(5, e)$:
$\boxed{S \to aBe \cdot}$

$12 = \tau(7, d)$:
$\boxed{S \to bBd \cdot}$

$13 = \tau(8, e)$:
$\boxed{S \to bAe \cdot}$

SLR(1) parsing
ANOTHER CASE STUDY

$$\begin{array}{lcl} S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\ A & \rightarrow & c \\ B & \rightarrow & c \end{array}$$



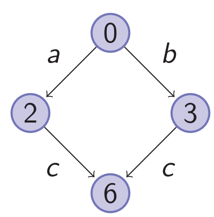
Is **not SLR(1)**

State $6 = \tau(2, c) = \tau(3, c)$ contains both $A \rightarrow c\,\cdot$ and $B \rightarrow c\,\cdot$

Hence, by $\mathrm{follow}(A) = \mathrm{follow}(B) = \{d, e\}$, there are r/r conflicts at both the entries $(6, d)$ and $(6, e)$ of the SLR(1) parsing table

SLR(1) parsing
ANOTHER CASE STUDY

$$\begin{array}{lcl} S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\ A & \rightarrow & c \\ B & \rightarrow & c \end{array}$$



Could not $\tau(2, c)$ and $\tau(3, c)$ be different states?
After all

- If we have read ac then we should reduce c to A only if the next symbol is d, and we should reduce c to B only if the next symbol is e
- Viceversa, if we have read bc then we should reduce c to A only if the next symbol is e, and we should reduce c to B only if the next symbol is d

LR(1)-items can make the difference