Introduction to Compiler Construction

Parsing: Bottom-up LR(1) Algorithm

Outline

1. Bottom-up Parsing
2. LR(1) Parsing
3. LR(1) Parse Tables
4. Conflicts

Bottom-up Parsing

Bottom-up Parsing

The bottom-up parser proceeds via a sequence of shifts and reductions, until the start symbol is on top of the stack and the input is just the terminator symbol `#`

Bottom-up Parsing

The bottom-up parser proceeds via a sequence of shifts and reductions, until the start symbol is on top of the stack and the input is just the terminator symbol `#`

Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \mathtt{id}$

Stack	Input	Action
	`id+id*id#`	shift
`id`	`+id*id#`	reduce 6
F	`+id*id#`	reduce 4
T	`+id*id#`	reduce 2
E	`+id*id#`	shift
$E+$	`id*id#`	shift
$E+\mathtt{id}$	`*id#`	reduce 6
$E+F$	`*id#`	reduce 4
$E+T$	`*id#`	shift
$E+T*$	`id#`	shift
$E+T*\mathtt{id}$	`#`	reduce 6
$E+T*F$	`#`	reduce 3
$E+T$	`#`	reduce 1
E	`#`	✓

Bottom-up Parsing

Bottom-up Parsing

The following questions arise

- How does the parser know when to shift and when to reduce?
- When reducing, how many symbols on top of the stack play a role in the reduction?
- Also, when reducing, by which rule does the parser make the reduction with?

Bottom-up Parsing

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

We call the sequence of symbols on top of the stack that are reduced to a single non-terminal at each reduction step the handle

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

We call the sequence of symbols on top of the stack that are reduced to a single non-terminal at each reduction step the handle

Formally, in a right-most derivation, $S \stackrel{*}{\Rightarrow} \alpha Y w \Rightarrow \alpha\beta w \stackrel{*}{\Rightarrow} uw$, a handle is a rule $Y ::= \beta$ and a position in $\alpha\beta w$ where β may be replaced by Y

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

We call the sequence of symbols on top of the stack that are reduced to a single non-terminal at each reduction step the handle

Formally, in a right-most derivation, $S \stackrel{*}{\Rightarrow} \alpha Y w \Rightarrow \alpha\beta w \stackrel{*}{\Rightarrow} uw$, a handle is a rule $Y ::= \beta$ and a position in $\alpha\beta w$ where β may be replaced by Y

When a handle appears on top of the stack

Stack	Input
$\alpha\beta$	w

we reduce that handle (β to Y in this case)

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

We call the sequence of symbols on top of the stack that are reduced to a single non-terminal at each reduction step the handle

Formally, in a right-most derivation, $S \stackrel{*}{\Rightarrow} \alpha Y w \Rightarrow \alpha\beta w \stackrel{*}{\Rightarrow} uw$, a handle is a rule $Y ::= \beta$ and a position in $\alpha\beta w$ where β may be replaced by Y

When a handle appears on top of the stack

Stack	Input
$\alpha\beta$	w

we reduce that handle (β to Y in this case)

If β is the sequence $X_1, X_2, \ldots, X_n$, then we call any subsequence, $X_1, X_2, \ldots, X_i$, for $i \leq n$ a viable prefix

Bottom-up Parsing

The stack configuration combined with the un-scanned input stream represents a sentential form in a right-most derivation of the input

We call the sequence of symbols on top of the stack that are reduced to a single non-terminal at each reduction step the handle

Formally, in a right-most derivation, $S \stackrel{*}{\Rightarrow} \alpha Y w \Rightarrow \alpha\beta w \stackrel{*}{\Rightarrow} uw$, a handle is a rule $Y ::= \beta$ and a position in $\alpha\beta w$ where β may be replaced by Y

When a handle appears on top of the stack

Stack	Input
$\alpha\beta$	w

we reduce that handle (β to Y in this case)

If β is the sequence $X_1, X_2, \ldots, X_n$, then we call any subsequence, $X_1, X_2, \ldots, X_i$, for $i \leq n$ a viable prefix

If there is not a handle on top of the stack and shifting an input token onto the stack results in a viable prefix, a shift is called for

LR(1) Parsing

LR(1) Parsing

The LR(1) parsing algorithm is a state machine with a pushdown stack, and is driven by two tables: Action and Goto

LR(1) Parsing

The LR(1) parsing algorithm is a state machine with a pushdown stack, and is driven by two tables: Action and Goto

A configuration of the parser is a pair, consisting of the state of the stack and the state of the input

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m$	$a_k a_{k+1} \ldots a_n$

where the s_i are states, the X_i are terminal or non-terminal symbols, and $a_k a_{k+1} \ldots a_n$ are the un-scanned input symbols

The LR(1) parsing algorithm is a state machine with a pushdown stack, and is driven by two tables: Action and Goto

A configuration of the parser is a pair, consisting of the state of the stack and the state of the input

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m$	$a_k a_{k+1} \ldots a_n$

where the s_i are states, the X_i are terminal or non-terminal symbols, and $a_k a_{k+1} \ldots a_n$ are the un-scanned input symbols

The configuration represents a right sentential form in a right-most derivation of the sequence $X_1 X_2 \ldots X_m a_k a_{k+1} \ldots a_n$

LR(1) Parsing · Example (Action and Goto Tables)

LR(1) Parsing · Example (Action and Goto Tables)

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

LR(1) Parsing · Example (Action and Goto Tables)

Input: Action and Goto tables and a sentence w
Output: a right-most derivation in reverse

1: Initially, the parser has the configuration,

Stack	Input
s_0	$a_1 a_2 \ldots a_n\#$

where $a_1 a_2 \ldots a_n$ is the input sentence

2: **repeat**

3: If Action$[s_m, a_k] = ss_i$, the parser executes a shift (the s stands for "shift") and goes into state s_i

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m a_k s_i$	$a_{k+1} \ldots a_n\#$

4: Otherwise, if Action$[s_m, a_k] = ri$ (the r stands for "reduce"), where i is the index of the rule $Y ::= X_j X_{j+1} \ldots X_m$, the parser replaces the symbols and states $X_j s_j X_{j+1} s_{j+1} \ldots X_m s_m$ by Ys, where $s =$ Goto$[s_{j-1}, Y]$, and outputs i

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \ldots X_{j-1} s_{j-1} Ys$	$a_{k+1} \ldots a_n\#$

5: Otherwise, if Action$[s_m, a_k] =$ accept, the parser halts successfully

6: Otherwise, if Action$[s_m, a_k] =$ error, the parser raises an error

7: **until** either the sentence is parsed or an error is raised

LR(1) Parsing · Example (parsing `id+id*id`)

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= `id`

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	`id+id*id#`	s5
0id5	`+id*id#`	r6

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	`id+id*id#`	s5
0id5	`+id*id#`	r6
0F3	`+id*id#`	r4

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T ∗ F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	∗	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	`id+id*id#`	s5
0id5	`+id*id#`	r6
0F3	`+id*id#`	r4
0T2	`+id*id#`	r2
0E1	`+id*id#`	s6
0E1+6	`id*id#`	s5
0E1+6id5	`*id#`	r6

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::=$ id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	`id+id*id#`	s5
0id5	`+id*id#`	r6
0F3	`+id*id#`	r4
0T2	`+id*id#`	r2
0E1	`+id*id#`	s6
0E1+6	`id*id#`	s5
0E1+6id5	`*id#`	r6
0E1+6F3	`*id#`	r4

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7
0E1+6T13*7	id#	s5

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7
0E1+6T13*7	id#	s5
0E1+6T13*7id5	#	r6

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7
0E1+6T13*7	id#	s5
0E1+6T13*7id5	#	r6
0E1+6T13*7F14	#	r3

LR(1) Parsing · Example (parsing `id+id*id`)

1. E ::= E + T
2. E ::= T
3. T ::= T * F
4. T ::= F
5. F ::= (E)
6. F ::= id

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7
0E1+6T13*7	id#	s5
0E1+6T13*7id5	#	r6
0E1+6T13*7F14	#	r3
0E1+6T13	#	r1

LR(1) Parsing · Example (parsing `id+id*id`)

1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Stack	Input	Action
0	id+id*id#	s5
0id5	+id*id#	r6
0F3	+id*id#	r4
0T2	+id*id#	r2
0E1	+id*id#	s6
0E1+6	id*id#	s5
0E1+6id5	*id#	r6
0E1+6F3	*id#	r4
0E1+6T13	*id#	s7
0E1+6T13*7	id#	s5
0E1+6T13*7id5	#	r6
0E1+6T13*7F14	#	r3
0E1+6T13	#	r1
0E1	#	✓

The LR(1) parsing tables, Action and Goto, for a grammar G are derived from a DFA for recognizing the possible handles for a parse in G

The LR(1) parsing tables, Action and Goto, for a grammar G are derived from a DFA for recognizing the possible handles for a parse in G

The DFA is constructed from the LR(1) canonical collection, a collection of sets of items (representing potential handles) of the form

$$[\mathrm{Y} ::= \alpha \cdot \beta, \mathtt{a}]$$

where $Y ::= \alpha\beta$ is a rule in P, α and β are (possibly empty) strings of symbols, and $\mathtt{a}$ is a lookahead symbol

The LR(1) parsing tables, Action and Goto, for a grammar G are derived from a DFA for recognizing the possible handles for a parse in G

The DFA is constructed from the LR(1) canonical collection, a collection of sets of items (representing potential handles) of the form

$$[\mathrm{Y} ::= \alpha \cdot \beta, \mathtt{a}]$$

where $Y ::= \alpha\beta$ is a rule in P, α and β are (possibly empty) strings of symbols, and $\mathtt{a}$ is a lookahead symbol

The $\cdot$ is a position marker that marks the top of the stack, indicating that we have parsed the α and still have the β ahead of us in satisfying the Y

The LR(1) parsing tables, Action and Goto, for a grammar G are derived from a DFA for recognizing the possible handles for a parse in G

The DFA is constructed from the LR(1) canonical collection, a collection of sets of items (representing potential handles) of the form

$$[\mathrm{Y} ::= \alpha \cdot \beta, \mathtt{a}]$$

where $Y ::= \alpha\beta$ is a rule in P, α and β are (possibly empty) strings of symbols, and $\mathtt{a}$ is a lookahead symbol

The $\cdot$ is a position marker that marks the top of the stack, indicating that we have parsed the α and still have the β ahead of us in satisfying the Y

The lookahead symbol $\mathtt{a}$ is a token that can follow Y (and so, $\alpha\beta$) in a legal right-most derivation of some sentence

The following item is called a possibility

$$[\mathrm{Y} ::= \cdot\ \alpha\ \beta,\ \mathtt{a}]$$

The following item is called a possibility

$$[\mathrm{Y} ::= \cdot\ \alpha\ \beta,\ \mathtt{a}]$$

The following item indicates that α has been parsed (and so is on the stack) but that there is still β to parse from the input

$$[\mathrm{Y} ::= \alpha\ \cdot\ \beta,\ \mathtt{a}]$$

The following item is called a possibility

$$[\mathrm{Y} ::= \cdot\ \alpha\ \beta,\ \mathtt{a}]$$

The following item indicates that α has been parsed (and so is on the stack) but that there is still β to parse from the input

$$[\mathrm{Y} ::= \alpha\ \cdot\ \beta,\ \mathtt{a}]$$

The following item indicates that the parser has successfully parsed $\alpha\beta$ in a context where $Y\mathtt{a}$ would be valid, and that the $\alpha\beta$ can be reduced to a Y

$$[\mathrm{Y} ::= \alpha\ \beta\ \cdot,\ \mathtt{a}]$$

The states in the DFA for recognizing viable prefixes and handles are constructed from items

The states in the DFA for recognizing viable prefixes and handles are constructed from items

We first augment our grammar G with an additional start symbol S' and an additional rule so as to yield an equivalent grammar G'

$$S' ::= S$$

The states in the DFA for recognizing viable prefixes and handles are constructed from items

We first augment our grammar G with an additional start symbol S' and an additional rule so as to yield an equivalent grammar G'

$$S' ::= S$$

Example (augmented arithmetic expression grammar)

$$
\begin{array}{lll}
0. & E' & ::= E \\
1. & E & ::= E \texttt{+} T \\
2. & E & ::= T \\
3. & T & ::= T \texttt{*} F \\
4. & T & ::= F \\
5. & F & ::= \texttt{(}E\texttt{)} \\
6. & F & ::= \texttt{id}
\end{array}
$$

The initial set, called kernel, representing the initial state in the DFA, will contain the LR(1) item

$$\{[S' ::= \cdot\ S, \texttt{\#}]\}$$

which says that parsing an S' means parsing an S from the input, after which point the next (and last) remaining token is the terminator #

The initial set, called kernel, representing the initial state in the DFA, will contain the LR(1) item

$$\{[S' ::= \cdot\ S,\ \texttt{\#}]\}$$

which says that parsing an S' means parsing an S from the input, after which point the next (and last) remaining token is the terminator `#`

The kernel may imply additional items, which are computed as the closure of the set

LR(1) Parse Tables · Closure of an Itemset

Input: itemset s
Output: closure(s)

$C \leftarrow \mathsf{Set}(s)$
repeat
  If C contains an item of the form

  $[Y ::= \alpha \cdot X\ \beta,\ \mathtt{a}]$,

  then add the item

  $[X ::= \cdot\ \gamma,\ \mathtt{b}]$

  to C for every rule $X ::= \gamma$ in P and for every token b in first($\beta\mathtt{a}$)
until no new items may be added
return C

Example

0. $E' ::= E$
1. $E ::= E \texttt{ + } T$
2. $E ::= T$
3. $T ::= T \texttt{ * } F$
4. $T ::= F$
5. $F ::= \texttt{(}E\texttt{)}$
6. $F ::= \texttt{id}$

Example

0. $E' ::= E$
1. $E ::= E \texttt{+} T$
2. $E ::= T$
3. $T ::= T \texttt{*} F$
4. $T ::= F$
5. $F ::= \texttt{(}E\texttt{)}$
6. $F ::= \texttt{id}$

closure($\{[E' ::= \cdot E, \texttt{\#}]\}$) yields

$$
\begin{aligned}
\{[E' &::= \cdot\ E, \texttt{\#}],\\
[E &::= \cdot\ E \texttt{+} T, \texttt{+/\#}],\\
[E &::= \cdot\ T, \texttt{+/\#}],\\
[T &::= \cdot\ T \texttt{*} F, \texttt{+/*/\#}],\\
[T &::= \cdot\ F, \texttt{+/*/\#}],\\
[F &::= \cdot\ \texttt{(}E\texttt{)}, \texttt{+/*/\#}],\\
[F &::= \cdot\ \texttt{id}, \texttt{+/*/\#}]\}
\end{aligned}
$$

which represents the initial state s_0 in the LR(1) canonical collection

LR(1) Parse Tables · goto(s, X)

For any item set s, and any symbol $X \in (T \cup N)$

$$\text{goto}(s, X) = \text{closure}(r),$$

where $r = \{[Y ::= \alpha X \cdot \beta, a] | [Y ::= \alpha \cdot X\beta, a]\}$

For any item set s, and any symbol $X \in (T \cup N)$

$$\text{goto}(s, X) = \text{closure}(r),$$

where $r = \{[Y ::= \alpha X \cdot \beta, a] | [Y ::= \alpha \cdot X\beta, a]\}$

Informally, to compute goto(s, X), take all items from s with a $\cdot$ before the X and move it after the X, and take the closure of that

LR(1) Parse Tables · goto(s, X)

LR(1) Parse Tables · goto(s, X)

Input: a state s, and a symbol $X \in T \cup N$
Output: the state goto(s, X)

```
r ← Set()
for [Y ::= α · Xβ, a] ∈ s do
    r.add([Y ::= αX · β, a])
end for
return closure(r)
```

LR(1) Parse Tables · goto(s, X)

Example

$$
\begin{array}{lll}
0. & E' & ::= E \\
1. & E & ::= E \texttt{+} T \\
2. & E & ::= T \\
3. & T & ::= T \texttt{*} F \\
4. & T & ::= F \\
5. & F & ::= \texttt{(}E\texttt{)} \\
6. & F & ::= \texttt{id}
\end{array}
$$

$$
\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \texttt{\#}], \\
&[E ::= \cdot\ E \texttt{+} T,\ \texttt{+/\#}], \\
&[E ::= \cdot\ T,\ \texttt{+/\#}], \\
&[T ::= \cdot\ T \texttt{*} F,\ \texttt{+/*/\#}], \\
&[T ::= \cdot\ F,\ \texttt{+/*/\#}], \\
&[F ::= \cdot\ \texttt{(}E\texttt{)},\ \texttt{+/*/\#}], \\
&[F ::= \cdot\ \texttt{id},\ \texttt{+/*/\#}]\}
\end{aligned}
$$

Example

0. $E' ::= E$
1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \mathtt{id}$

$$
\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \#],\\
&[E ::= \cdot\ E + T,\ +/\#],\\
&[E ::= \cdot\ T,\ +/\#],\\
&[T ::= \cdot\ T * F,\ +/*/\#],\\
&[T ::= \cdot\ F,\ +/*/\#],\\
&[F ::= \cdot\ (E),\ +/*/\#],\\
&[F ::= \cdot\ \mathtt{id},\ +/*/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{goto}(s_0, E) = s_1 = \{&[E' ::= E\ \cdot,\ \#],\\
&[E ::= E\ \cdot + T,\ +/\#]\}
\end{aligned}
$$

Example

0. $E' ::= E$
1. $E ::= E + T$
2. $E ::= T$
3. $T ::= T * F$
4. $T ::= F$
5. $F ::= (E)$
6. $F ::= \texttt{id}$

$$
\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \#],\\
&[E ::= \cdot\ E + T,\ +/\#],\\
&[E ::= \cdot\ T,\ +/\#],\\
&[T ::= \cdot\ T * F,\ +/*/\#],\\
&[T ::= \cdot\ F,\ +/*/\#],\\
&[F ::= \cdot\ (E),\ +/*/\#],\\
&[F ::= \cdot\ \texttt{id},\ +/*/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ E) = s_1 = \{&[E' ::= E\ \cdot,\ \#],\\
&[E ::= E\ \cdot + T,\ +/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ T) = s_2 = \{&[E ::= T\ \cdot,\ +/\#],\\
&[T ::= T\ \cdot * F,\ +/*/\#]\}
\end{aligned}
$$

Example

$$
\begin{aligned}
&0.\ E' ::= E\\
&1.\ E ::= E + T\\
&2.\ E ::= T\\
&3.\ T ::= T * F\\
&4.\ T ::= F\\
&5.\ F ::= (E)\\
&6.\ F ::= \texttt{id}
\end{aligned}
$$

$$
\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \#],\\
&[E ::= \cdot\ E + T,\ +/\#],\\
&[E ::= \cdot\ T,\ +/\#],\\
&[T ::= \cdot\ T * F,\ +/*/\#],\\
&[T ::= \cdot\ F,\ +/*/\#],\\
&[F ::= \cdot\ (E),\ +/*/\#],\\
&[F ::= \cdot\ \texttt{id},\ +/*/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ E) = s_1 = \{&[E' ::= E\ \cdot,\ \#],\\
&[E ::= E\ \cdot + T,\ +/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ T) = s_2 = \{&[E ::= T\ \cdot,\ +/\#],\\
&[T ::= T\ \cdot * F,\ +/*/\#]\}
\end{aligned}
$$

$$
\text{goto}(s_0,\ F) = s_3 = \{[T ::= F\ \cdot,\ +/*/\#]\}
$$

Example

$$
\begin{aligned}
&0.\ E' ::= E \\
&1.\ E ::= E + T \\
&2.\ E ::= T \\
&3.\ T ::= T * F \\
&4.\ T ::= F \\
&5.\ F ::= (E) \\
&6.\ F ::= \texttt{id}
\end{aligned}
$$

$$
\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \#], \\
&[E ::= \cdot\ E + T,\ +/\#], \\
&[E ::= \cdot\ T,\ +/\#], \\
&[T ::= \cdot\ T * F,\ +/*/\#], \\
&[T ::= \cdot\ F,\ +/*/\#], \\
&[F ::= \cdot\ (E),\ +/*/\#], \\
&[F ::= \cdot\ \texttt{id},\ +/*/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ E) = s_1 = \{&[E' ::= E\ \cdot,\ \#], \\
&[E ::= E\ \cdot + T,\ +/\#]\}
\end{aligned}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ T) = s_2 = \{&[E ::= T\ \cdot,\ +/\#], \\
&[T ::= T\ \cdot * F,\ +/*/\#]\}
\end{aligned}
$$

$$
\text{goto}(s_0,\ F) = s_3 = \{[T ::= F\ \cdot,\ +/*/\#]\}
$$

$$
\begin{aligned}
\text{goto}(s_0,\ () = s_4 = \{&[F ::= (\cdot\ E),\ +/*/\#], \\
&[E ::= \cdot\ E + T,\ +/)], \\
&[E ::= \cdot\ T,\ +/)], \\
&[T ::= \cdot\ T * F,\ +/*/)], \\
&[T ::= \cdot\ F,\ +/*/)], \\
&[F ::= \cdot\ (E),\ +/*/)], \\
&[F ::= \cdot\ \texttt{id},\ +/*/)]\}
\end{aligned}
$$

Example

$$\begin{aligned}
&0.\ E' ::= E\\
&1.\ E ::= E + T\\
&2.\ E ::= T\\
&3.\ T ::= T * F\\
&4.\ T ::= F\\
&5.\ F ::= (E)\\
&6.\ F ::= \mathtt{id}
\end{aligned}$$

$$\begin{aligned}
s_0 = \{&[E' ::= \cdot\ E,\ \#],\\
&[E ::= \cdot\ E + T,\ +/\#],\\
&[E ::= \cdot\ T,\ +/\#],\\
&[T ::= \cdot\ T * F,\ +/*/\#],\\
&[T ::= \cdot\ F,\ +/*/\#],\\
&[F ::= \cdot\ (E),\ +/*/\#],\\
&[F ::= \cdot\ \mathtt{id},\ +/*/\#]\}
\end{aligned}$$

$$\begin{aligned}
\text{goto}(s_0,\ E) = s_1 = \{&[E' ::= E\ \cdot,\ \#],\\
&[E ::= E\ \cdot + T,\ +/\#]\}
\end{aligned}$$

$$\begin{aligned}
\text{goto}(s_0,\ T) = s_2 = \{&[E ::= T\ \cdot,\ +/\#],\\
&[T ::= T\ \cdot * F,\ +/*/\#]\}
\end{aligned}$$

$$\text{goto}(s_0,\ F) = s_3 = \{[T ::= F\ \cdot,\ +/*/\#]\}$$

$$\begin{aligned}
\text{goto}(s_0,\ () = s_4 = \{&[F ::= (\cdot\ E),\ +/*/\#],\\
&[E ::= \cdot\ E + T,\ +/)],\\
&[E ::= \cdot\ T,\ +/)],\\
&[T ::= \cdot\ T * F,\ +/*/)],\\
&[T ::= \cdot\ F,\ +/*/)],\\
&[F ::= \cdot\ (E),\ +/*/)],\\
&[F ::= \cdot\ \mathtt{id},\ +/*/)]\}
\end{aligned}$$

$$\text{goto}(s_0,\ \mathtt{id}) = s_5 = \{[F ::= \mathtt{id}\ \cdot,\ +/*/\#]\}$$

LR(1) Parse Tables · LR(1) Canonical Collection

Input: a context-free grammar $G = (N, T, S, P)$

Output: the canonical LR(1) collection of states $\mathcal{C} = \{s_0, s_1, \ldots, s_n\}$

Define an augmented grammar G' which is G with the added non-terminal S' and added production rule $S' ::= S$
$s_0 \leftarrow \text{closure}(\{[S' ::= \cdot S, \#]\})$
$\mathcal{C} \leftarrow \text{Set}(s_0)$
repeat
    for $s \in \mathcal{C}$ **do**
        for $X \in T \cup N$ **do**
            if $\text{goto}(s, X) \neq \emptyset$ and $\text{goto}(s, X) \notin \mathcal{C}$ **then**
                $\mathcal{C}.\text{add}(\text{goto}(s, X))$
            end if
        end for
    end for
until no new states are added to $\mathcal{C}$

Example (the LR(1) canonical collection for the arithmetic expression grammar)

$s_0 = \{[E' ::= \cdot E, \#], [E ::= \cdot E{+}T, +/\#], [E ::= \cdot T, +/\#], [T ::= \cdot T{*}F, +/{*}/\#], [T ::= \cdot F, +/{*}/\#], [F ::= \cdot(E), +/{*}/\#], [F ::= \cdot\mathtt{id}, +/{*}/\#]\}$

$\mathrm{goto}(s_0, E) = \{[E' ::= E\cdot, \#], [E ::= E \cdot {+}T, +/\#]\} = s_1$
$\mathrm{goto}(s_0, T) = \{[E ::= T\cdot, +/\#], [T ::= T \cdot {*}F, +/{*}/\#], \} = s_2$
$\mathrm{goto}(s_0, F) = \{[T ::= F\cdot, +/{*}/\#]\} = s_3$
$\mathrm{goto}(s_0, () = \{[F ::= (\cdot E), +/{*}/\#], [E ::= \cdot E{+}T, +/)], [E ::= \cdot T, +/)], [T ::= \cdot T{*}F, +/{*}/)], [T ::= \cdot F, +/{*}/)], [F ::= \cdot(E), +/{*}/)], [F ::= \cdot\mathtt{id}, +/{*}/)]\} = s_4$
$\mathrm{goto}(s_0, \mathtt{id}) = \{[F ::= \mathtt{id}\cdot, +/{*}/\#]\} = s_5$

$\mathrm{goto}(s_1, +) = \{[E ::= E{+} \cdot T, +/\#], [T ::= \cdot T{*}F, +/{*}/\#], [T ::= \cdot F, +/{*}/\#], [F ::= \cdot(E), +/{*}/\#], [F ::= \cdot\mathtt{id}, +/{*}/\#]\} = s_6$

$\mathrm{goto}(s_2, *) = \{[T ::= T{*} \cdot F, +/{*}/\#], [F ::= \cdot(E), +/{*}/\#], [F ::= \cdot\mathtt{id}, +/{*}/\#]\} = s_7$

$\mathrm{goto}(s_4, E) = \{[F ::= (E \cdot), +/{*}/\#], [E ::= E \cdot {+}T, +/)]\} = s_8$
$\mathrm{goto}(s_4, T) = \{[E ::= T\cdot, +/)], [T ::= T \cdot {*}F, +/{*}/)]\} = s_9$
$\mathrm{goto}(s_4, F) = \{[T ::= F\cdot, +/{*}/)]\} = s_{10}$
$\mathrm{goto}(s_4, () = \{[F ::= (\cdot E), +/{*}/)], [E ::= \cdot E{+}T, +/)], [E ::= \cdot T, +/)], [T ::= \cdot T{*}F, +/{*}/)], [T ::= \cdot F, +/{*}/)], [F ::= \cdot(E), +/{*}/)], [F ::= \cdot\mathtt{id}, +/{*}/)]\} = s_{11}$
$\mathrm{goto}(s_4, \mathtt{id}) = \{[F ::= \mathtt{id}\cdot, +/{*}/)]\} = s_{12}$

$\mathrm{goto}(s_6, T) = \{[E ::= E{+}T\cdot, +/\#], [T ::= T \cdot {*}F, +/{*}/\#]\} = s_{13}$
$\mathrm{goto}(s_6, F) = s_3$
$\mathrm{goto}(s_6, () = s_4$
$\mathrm{goto}(s_6, \mathtt{id}) = s_5$

$\mathrm{goto}(s_7, F) = \{[T ::= T{*}F\cdot, +/{*}/\#]\} = s_{14}$
$\mathrm{goto}(s_7, () = s_4$

$\mathrm{goto}(s_7, \mathtt{id}) = s_5$

$\text{goto}(s_8,)) = \{[F ::= (E)\cdot, +/*/\#]\} = s_{15}$
$\text{goto}(s_8, +) = \{[E ::= E+\cdot T, +/)], [T ::= \cdot T*F, +/*/)], [T ::= \cdot F, +/*/)], [F ::= \cdot(E), +/*/)], [F ::= \cdot\texttt{id}, +/*/)]\} = s_{16}$

$\text{goto}(s_9, *) = \{[T ::= T*\cdot F, +/*/)], [F ::= \cdot(E), +/*/)], [F ::= \cdot\texttt{id}, +/*/)]\} = s_{17}$

$\text{goto}(s_{11}, E) = \{[F ::= (E\cdot), +/*/)], [E ::= E\cdot +T, +/)]\} = s_{18}$
$\text{goto}(s_{11}, T) = s_9$
$\text{goto}(s_{11}, F) = s_{10}$
$\text{goto}(s_{11}, () = s_{11}$
$\text{goto}(s_{11}, \texttt{id}) = s_{12}$

$\text{goto}(s_{13}, *) = s_7$

$\text{goto}(s_{16}, T) = \{[E ::= E+T\cdot, +/)][T ::= T\cdot *F, +/*/)]\} = s_{19}$
$\text{goto}(s_{16}, F) = s_{10}$
$\text{goto}(s_{16}, () = s_{11}$
$\text{goto}(s_{16}, \texttt{id}) = s_{12}$

$\text{goto}(s_{17}, F) = \{[T ::= T*F\cdot, +/*/)]\} = s_{20}$
$\text{goto}(s_{17}, () = s_{11}$
$\text{goto}(s_{17}, \texttt{id}) = s_{12}$

$\text{goto}(s_{18}, () = \{[F ::= (E)\cdot, +/*/)], \} = s_{21}$
$\text{goto}(s_{18}, +) = s_{16}$

$\text{goto}(s_{19}, *) = s_{17}$

LR(1) Parse Tables

LR(1) Parse Tables

Input: a context-free grammar $G = (N, T, S, P)$
Output: the LR(1) tables Action and Goto

1. Compute the LR(1) canonical collection $\mathcal{C} = \{s_0, s_1, \ldots, s_n\}$
2. The Action table is constructed as follows:
   a For each transition, $\mathrm{goto}(s_i, \mathtt{a}) = s_j$, where a is a terminal, set $\mathrm{Action}[i, \mathtt{a}] = sj$
   b If the item set s_k contains the item $[S' ::= S\cdot, \mathtt{\#}]$, set $\mathrm{Action}[k, \mathtt{\#}] =$ accept
   c For all item sets s_i, if s_i contains an item of the form $[Y ::= \alpha\cdot, \mathtt{a}]$, set $\mathrm{Action}[i, \mathtt{a}] = rp$, where p is the number of the rule $Y ::= \alpha$
   d All undefined entries in Action are set to error
3. The Goto table is constructed as follows:
   a For each transition, $\mathrm{goto}(s_i, Y) = s_j$, where Y is a non-terminal, set $\mathrm{Goto}[i, Y] = j$
   b All undefined entries in Goto are set to error

LR(1) Parse Tables

LR(1) Parse Tables

Example (Action and Goto tables for the arithmetic expression grammar)

	Action						Goto		
	+	*	(	)	id	#	E	T	F
0			s4		s5		1	2	3
1	s6					✓			
2	r2	s7				r2			
3	r4	r4				r4			
4			s11		s12		8	9	10
5	r6	r6				r6			
6			s4		s5			13	3
7			s4		s5				14
8	s16			s15					
9	r2	s17		r2					
10	r4	r4		r4					
11			s11		s12		18	9	10
12	r6	r6		r6					
13	r1	s7				r1			
14	r3	r3				r3			
15	r5	r5				r5			
16			s11		s12			19	10
17			s11		s12				20
18	s16			s21					
19	r1	s17		r1					
20	r3	r3		r3					
21	r5	r5		r5					

Conflicts

Conflicts

There are two different kinds of conflicts possible for an entry in the Action table

Conflicts

There are two different kinds of conflicts possible for an entry in the Action table

The shift-reduce conflict can occur when there are items of the forms

$[Y ::= \alpha\ \cdot, \texttt{a}]$ and
$[Y ::= \alpha\ \cdot \texttt{a} \beta, \texttt{b}]$

Conflicts

There are two different kinds of conflicts possible for an entry in the Action table

The shift-reduce conflict can occur when there are items of the forms

$[Y ::= \alpha\ \cdot, \texttt{a}]$ and
$[Y ::= \alpha\ \cdot\texttt{a}\beta, \texttt{b}]$

Example (the dangling else problem)

$S ::= \texttt{if (}E\texttt{)}\ S$
$S ::= \texttt{if (}E\texttt{)}\ S\ \texttt{else}\ S$

Most parser generators that are based on LR grammars favor a shift of the $\texttt{else}$ over a reduce of the $\texttt{if (}E\texttt{)}\ S$ to an S

Conflicts

Conflicts

The reduce-reduce conflict can happen when we have a state containing two items of the form

$$[X ::= \alpha \cdot, \texttt{a}]$$
$$[Y ::= \beta \cdot, \texttt{a}]$$