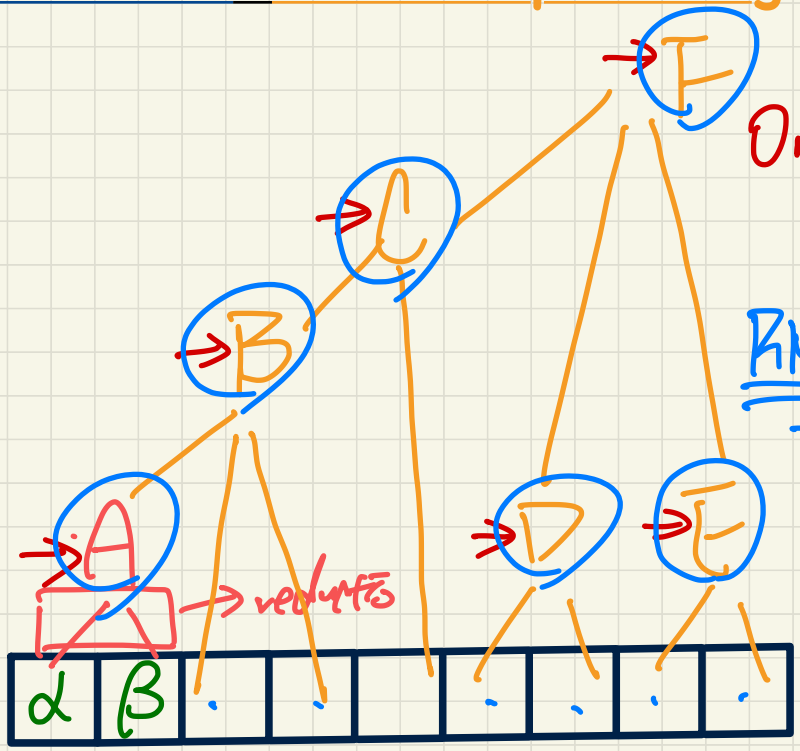
Discovering Derivations: Bottom-Up Parsing

production rule

$A \rightarrow \alpha\beta$

scanner



Order of reductions:

A B C D E F.

RMD:

F E D C B A

Bottom-Up Parsing: Algorithm

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word + state */
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ''shift to s'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

State	Action Table			Goto Table	
	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Bottom-Up Parsing: Discovering Rightmost Derivations (1)

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ''shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

1	Goal → List
2	List → List Pair
3	| Pair
4	Pair → (Pair)
5	| ( )

State	Action Table: eof	(	)	Goto Table: List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Parse: ( )

word: "(" ")" eof

state: 0 3 6 7 8 5 1

handle

1. List

trace

Bottom-Up Parsing: Discovering Rightmost Derivations (2)

Parse: (( )) ( )

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success  or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ``accept'' then
      succeed()
    elseif act = ``reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ``shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

	Action Table			Goto Table	
State	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Bottom-Up Parsing: Discovering Rightmost Derivations (3)

```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success  or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(0) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce based on A → β'' then
      trace.pop() 2 × |β| times /* word +
      state := trace.top()
      trace.push(A)
      next := Goto[state, A]
      trace.push(next)
    elseif act = ''shift to s_i'' then
      trace.push(word)
      trace.push(i)
      word := NextWord()
    else
      fail()
```

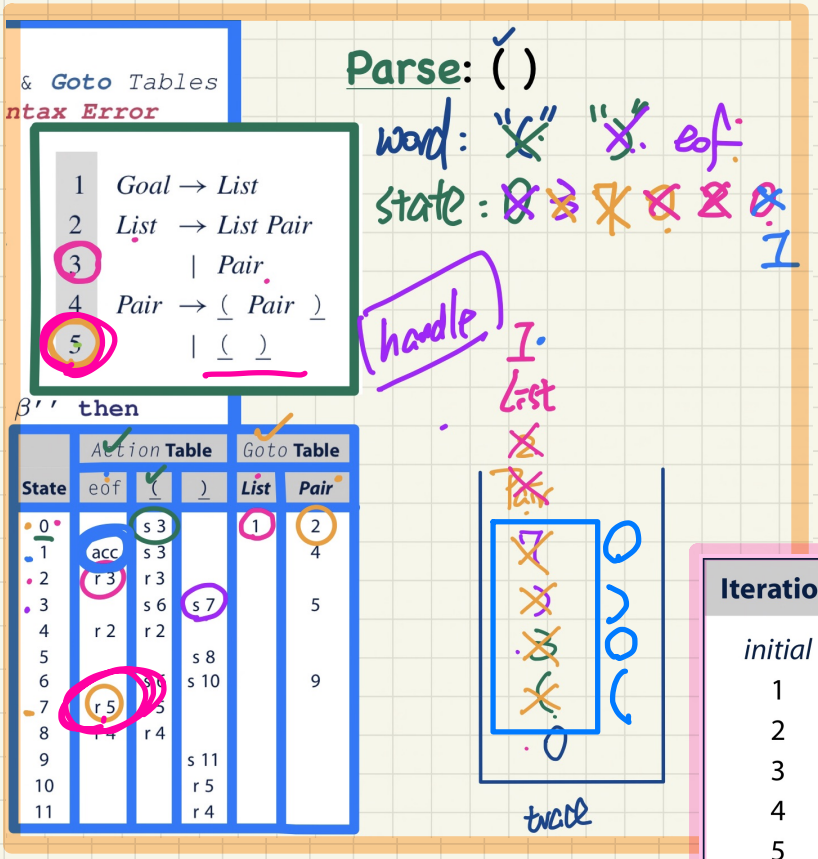
Parse: ( ))

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

	Action Table			Goto Table	
State	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

Bottom-Up Parsing: Handles

A handle denotes a parser's state that's ready for reduction.



Iteration	State	word	Stack	Handle	Action
initial	—	(	$ 0	*none*	—
1	0	(	$ 0	*none*	*shift 3*
2	3	)	$ 0 (3	*none*	*shift 7*
3	7	eof	$ 0 (3) 7	( )	*reduce 5*
4	2	eof	$ 0 *Pair* 2	*Pair*	*reduce 3*
5	1	eof	$ 0 *List* 1	*List*	*accept*

state ready for reduction

Bottom-Up Parsing: Right-Most Derivation

Parse: (( )) ( )

The BUP process corresponds to the reverse of a RMD.

Iteration	State	word	Stack	Handle	Action
initial	—	(	$ 0	— none —	—
1	0	(	$ 0	— none —	*shift 3*
2	3	(	$ 0 (3	— none —	*shift 6*
3	6	)	$ 0 (3 (6	— none —	*shift 10*
4	10	)	$ 0 (3 (6) 10	( )	*reduce 5*
5	5	)	$ 0 (3 *Pair* 5	— none —	*shift 8*
6	8	(	$ 0 (3 *Pair* 5) 8	(*Pair*)	*reduce 4*
7	2	(	$ 0 *Pair* 2	*Pair*	*reduce 3*
8	1	(	$ 0 *List* 1	— none —	*shift 3*
9	3	)	$ 0 *List* 1 (3	— none —	*shift 7*
10	7	eof	$ 0 *List* 1 (3) 7	( )	*reduce 5*
11	4	eof	$ 0 *List* 1 *Pair* 4	*List Pair*	*reduce 2*
12	1	eof	$ 0 *List* 1	*List*	*accept*

1 *Goal* → *List*
2 *List* → *List Pair*
3 | *Pair*
4 *Pair* → (*Pair*)
5 | ( )

Order of reduction: 1 – 5
Order of RMD: reverse!

LR(1) Items: Definition

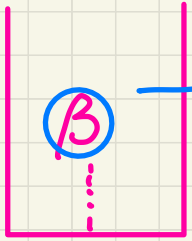
$$[A \to \beta \bullet \gamma, a]$$

production rule $A \to \beta\gamma$

possible states of parser

current top of stack

look-ahead $\in$ Follow(A)



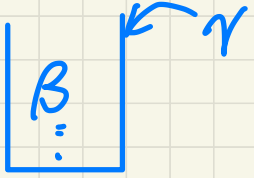
- we have already recognize β
- once we recognize γ
  - ↳ in a handle ready for reducing into A

LR(1) Items: Scenarios

Possibility: $[A \rightarrow \bullet\ \beta\ \gamma,\ a]$

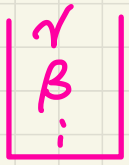
↳ initial state of parsing towards reduction to A

Partial Completion: $[A \rightarrow \beta \bullet \gamma,\ a]$



↳ already recognized β
still expecting to recognize γ

Completion: $[A \rightarrow \beta\ \gamma \bullet,\ a]$



$\in$ Follow(A)
if word matches a, reduce to A

LR(1) Items: Exercise (1.1a)

1	*Goal*	→	*List*
2	*List*	→	*List Pair*
3		|	*Pair*
4	*Pair*	→	(*Pair*)
5		|	( )

Q. LR(1) item denoting the initial state of parsing?

[Goal → • List , eof] ← Follow(Goal) = {eof}

Q. LR(1) item denoting the desired final state of parsing?

not necessarily = the final state ✗ [Pair → ( ) • ,] [Goal → List • , eof]

LR(1) Items: Exercise (1.1b)

Q. Derive all LR(1) items for the production rule $A \rightarrow \beta\gamma$

- union
- set comprehension
- floating "point"

D_1 : floating positions of •

→ $A \rightarrow \bullet\beta\gamma$

$A \rightarrow \beta\bullet\gamma$

$A \rightarrow \beta\gamma\bullet$

D_2 : Follow(A)

$\{[A \rightarrow \bullet\beta\gamma, a] \mid a \in \text{Follow}(A)\}$

$\cup$

$\{[A \rightarrow \beta\bullet\gamma, a] \mid a \in \text{Follow}(A)\}$

$\cup$

$\{[A \rightarrow \beta\gamma\bullet, a] \mid a \in \text{Follow}(A)\}$

LR(1) Items: Exercise (1.2)

1	*Goal*	→	*List*	
2	*List*	→	*List Pair*	
3		|	*Pair*	
4	*Pair*	→	(*Pair*)	
5		|	( )	

How many LR(1) items?
- possible floating point positions: 4
- cardinality of Follow(Pair) | Follow(Pair)| : 3

12.

Q. Derive all LR(1) items for the production rule Pair → (Pair)

FOLLOW(*List*) = {eof, (} **FOLLOW**(*Pair*) = {eof, (,)}

[Pair → • (Pair), eof] [Pair → (• Pair), eof] [Pair → (Pair •), eof] [Pair → (Pair) •, eof]
[Pair → • (Pair), ( ] [Pair → (• Pair), ( ] [Pair → (Pair •), ( ] [Pair → (Pair) •, ( ]
[Pair → • (Pair),)] [Pair → (• Pair),)] [Pair → (Pair •),)] [Pair → (Pair) •,)]

LR(1) Items: Exercise (1.3)

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow \underline{(}\ Pair\ \underline{)}$
5	$\mid \underline{(}\ \underline{)}$

FOLLOW(*List*) = {eof, (} **FOLLOW**(*Pair*) = {eof, (,)}

$[Goal \rightarrow \bullet\ List, \texttt{eof}]$

$[Goal \rightarrow List\ \bullet, \texttt{eof}]$

$[List \rightarrow \bullet\ List\ Pair, \texttt{eof}]$ $[List \rightarrow \bullet\ List\ Pair, \underline{(}\,]$
$[List \rightarrow List \bullet Pair, \texttt{eof}]$ $[List \rightarrow List \bullet Pair, \underline{(}\,]$
$[List \rightarrow List\ Pair \bullet, \texttt{eof}]$ $[List \rightarrow List\ Pair \bullet, \underline{(}\,]$

$[List \rightarrow \bullet\ Pair, \texttt{eof}\,]$ $[List \rightarrow \bullet\ Pair, \underline{(}\,]$
$[List \rightarrow Pair \bullet, \texttt{eof}\,]$ $[List \rightarrow Pair \bullet, \underline{(}\,]$

$[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \texttt{eof}\,]$ $[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \underline{)}]$ $[Pair \rightarrow \bullet\ \underline{(}\ Pair\ \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \texttt{eof}\,]$ $[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \underline{)}]$ $[Pair \rightarrow \underline{(} \bullet Pair\ \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(}\ Pair \bullet\ \underline{)}, \texttt{eof}\,]$ $[Pair \rightarrow \underline{(}\ Pair \bullet\ \underline{)}, \underline{)}]$ $[Pair \rightarrow \underline{(}\ Pair \bullet\ \underline{)}, \underline{(}]$
$[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \texttt{eof}\,]$ $[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \underline{)}]$ $[Pair \rightarrow \underline{(}\ Pair\ \underline{)} \bullet, \underline{(}]$

$[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \texttt{eof}]$ $[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \underline{(}]$ $[Pair \rightarrow \bullet\ \underline{(}\ \underline{)}, \underline{)}]$
$[Pair \rightarrow \underline{(} \bullet \underline{)}, \texttt{eof}]$ $[Pair \rightarrow \underline{(} \bullet \underline{)}, \underline{(}]$ $[Pair \rightarrow \underline{(} \bullet \underline{)}, \underline{)}]$
$[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \texttt{eof}]$ $[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \underline{(}]$ $[Pair \rightarrow \underline{(}\ \underline{)} \bullet, \underline{)}]$

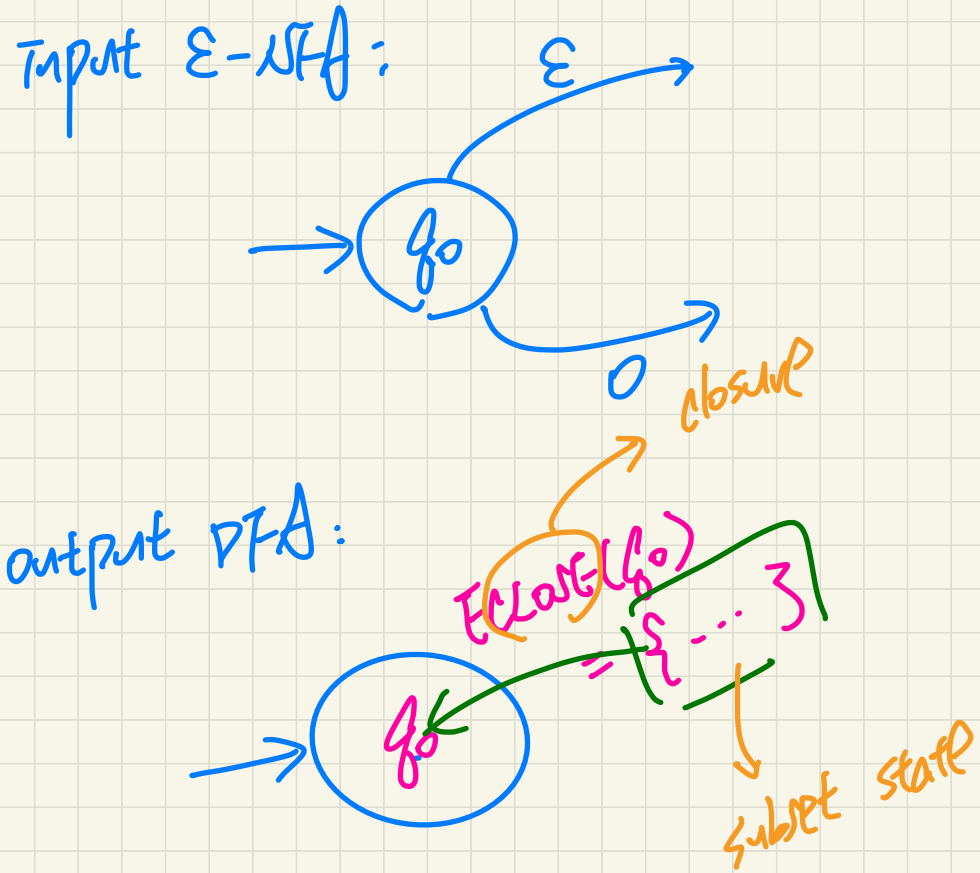
LR(1) Items: Exercise (2)

0	*Goal*	→	*Expr*	6	*Term′*	→	x *Factor Term′*
1	*Expr*	→	*Term Expr′*	7		|	÷ *Factor Term′*
2	*Expr′*	→	+ *Term Expr′*	8		|	ϵ
3		|	- *Term Expr′*	9	*Factor*	→	(*Expr*)
4		|	ϵ	10		|	num
5	*Term*	→	*Factor Term′*	11		|	name

Q. Derive all LR(1) items for the the above grammar.

FOLLOW Set

	Expr	*Expr′*	*Term*	*Term′*	*Factor*
FOLLOW	eof,)	eof,)	eof, +, -,)	eof, +, -,)	eof, +, -, x, ÷,)


Input ε-NFA:
ε
q0
0
output DFA:
closure
ECLOSE(q0)
{ ... }
q0
subset state

CC Construction: closure

```
1  ALGORITHM: closure
2    INPUT: CFG G = (V, Σ, R, S), a set s of LR(1) items
3    OUTPUT: a set of LR(1) items
4  PROCEDURE:
5    lastS := ∅
6    while (lastS ≠ s):
7      lastS := s
8      for [A → ··· • C δ, a] ∈ s:
9        for C → γ ∈ R:
10         for b ∈ FIRST(δa):
11           s := s ∪ { [C → •γ, b] }
12   return s
```

keep growing the output set s until nothing new can be added.

∀ the "follow" of C

1. What has been recognized? ···
2. What's expected to be recognized next? C

all alternatives starting with C

to C.

b ∈ FIRST(δa)

Q. Why not ε ∈ FIRST(δ)?

b ∈ FIRST(δ)

∴ δ might be nullable

Analogy: ε-NFA to DFA

Subset construction (with ***lazy evaluation*** and *epsilon closures*) produces a ***DFA*** transition table.

	$d \in 0..9$	$s \in \{+,-\}$	.
$\{q_0, q_1\}$	$\{q_1, q_4\}$	$\{q_1\}$	$\{q_2\}$
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\varnothing$	$\{q_2, q_3, q_5\}$
$\{q_1\}$	$\{q_1, q_4\}$	$\varnothing$	$\{q_2\}$
$\{q_2\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$
$\{q_2, q_3, q_5\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$
$\{q_3, q_5\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$

starting set

For example, $\delta(\{q_0, q_1\}, d)$ is calculated as follows: $[d \in 0..9]$

$$\cup\{\text{ECLOSE}(q) \mid q \in \delta(q_0, d) \cup \delta(q_1, d)\}$$

$[C \to \bullet\gamma, b]$

new LR(1) item to be added to the closure.

CC Construction: CC_0

set of subset states → a set of LR(1) items

Calculate CC_0 of the following grammar.

Hint: Closure of the singleton set containing the parser's initial state.

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

```
1  ALGORITHM: closure
2    INPUT: CFG G = (V, Σ, R, S), a set s of LR(1) items
3    OUTPUT: a set of LR(1) items
4  PROCEDURE:
5    lastS := ∅
6    while (lastS ≠ s):
7      lastS := s
8      for [A → ··· • C δ, a] ∈ s:
9        for C → γ ∈ R:
10         for b ∈ FIRST(δa):
11           s := s ∪ { [C → •γ, b] }
12   return s
```

initial subset state of the parser

ε-NFA → q_0 → DFA → ECLOSE(q_0) → CC_0

parser's initial state:

{ [Goal → • List, eof] }

= input to closure

CC Construction: CC_0 Step 1

(0) [Goal → • List, eof] initial parser state

Hint 1. How is [A → β • C δ, a] instantiated?

Goal ε List ε eof

Hint 2. What are $C \to \gamma \in R$?

→ List → List Pair List → Pair

Hint 3. FIRST(δa) = FIRST(ε eof) = FIRST(eof) = { eof }

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

( ) (( )) (( ))( )

Two new LR(1) items:

1. [List → • List Pair, eof]
2. [List → • Pair, eof]

How should s be extended?

```
for [A → ··· • C δ, a] ∈ s:
  for C → γ ∈ R:
    for b ∈ FIRST(δa):
      s := s ∪ { [ C → • γ, b] }
```

$CC_0 = \{$ [*Goal* → • *List*, eof] [*List* → • *List Pair*, eof] [*List* → • *List Pair*, (] [*List* → • *Pair*, eof] [*List* → • *Pair*, (] [*Pair* → • (*Pair*), eof] [*Pair* → • (*Pair*),(] [*Pair* → • ( ), eof] [*Pair* → • ( ),(] $\}$

CC Construction: CC_0 Step 2

(0) [Goal → • List, eof]

(1) [List → • List Pair, eof]

(2) [List → • Pair, eof]

Hint 1. How is $[A \to \beta \bullet C\ \delta, a]$ instantiated?

Hint 2. What are $C \to \gamma \in R$?

Hint 3. FIRST(δa) =

How should s be extended?

```
for [A → ··· • C δ, a] ∈ s:
  for C → γ ∈ R:
    for b ∈ FIRST(δa):
      s := s ∪ { [C → •γ, b] }
```

1 Goal → List
2 List → List Pair
3 | Pair
4 Pair → (Pair)
5 | ( )

$CC_0 = \{$ [Goal → • List, eof], [List → • List Pair, eof], [List → • List Pair, (], [List → • Pair, eof], [List → • Pair, (], [Pair → • (Pair), eof], [Pair → • (Pair), (], [Pair → • ( ), eof], [Pair → • ( ), (] $\}$

CC Construction: CC_0 Step 3

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

(0) [Goal → • List, eof]

(1) [List → • List Pair, eof]

(2) [List → • Pair, eof]

(3) [Pair → • (Pair), eof]

(4) [Pair → • ( ), eof]

Hint 1. How is $[A \rightarrow \beta \bullet C\ \delta, a]$ instantiated?

Hint 2. What are $C \rightarrow \gamma \in R$?

Hint 3. FIRST(δa) = FIRST(Pair eof) = { (}

How should s be extended?

```
for [A → ··· • C δ, a] ∈ s:
  for C → γ ∈ R:
    for b ∈ FIRST(δa):
      s := s ∪ { [ C → •γ, b] }
```

$$CC_0 = \left\{ \begin{array}{lll} [Goal \rightarrow \bullet List, \texttt{eof}] & [List \rightarrow \bullet List\ Pair, \texttt{eof}] & [List \rightarrow \bullet List\ Pair, (] \\ [List \rightarrow \bullet Pair, \texttt{eof}] & [List \rightarrow \bullet Pair, (] & [Pair \rightarrow \bullet (\ Pair\ ), \texttt{eof}] \\ [Pair \rightarrow \bullet (\ Pair\ ), (] & [Pair \rightarrow \bullet (\ ), \texttt{eof}] & [Pair \rightarrow \bullet (\ ), (] \end{array} \right\}$$

CC Construction: CC_0 Step 4

(0) [Goal → • List, eof]
(1) [List → • List Pair, eof]
(2) [List → • Pair, eof]
(3) [Pair → • (Pair), eof]
(4) [Pair → • ( ), eof]
(5) [List → • List Pair, ( ]
(6) [List → • Pair, ( ]

1	*Goal* → *List*
2	*List* → *List Pair*
3	| *Pair*
4	*Pair* → (*Pair*)
5	| ( )

- Hint 1. How is $[A \to \beta \bullet C\ \delta, a]$ instantiated?
- Hint 2. What are $C \to \gamma \in R$?
- Hint 3. FIRST(δa) =

List ε Pair ε (

FIRST(ε ( ) = { (}

How should s be extended?

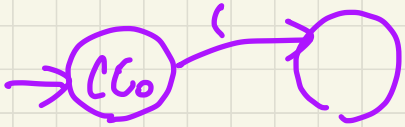
```
for [A → ··· • C δ, a] ∈ s:
  for C → γ ∈ R:
    for b ∈ FIRST(δa):
      s := s ∪ { [C → •γ, b] }
```

Two additional LR(1) items:

1. [Pair → • (Pair), ( ]
2. [Pair → • ( ), ( ]

$CC_0 = \{$
[Goal → • List, eof] [List → • List Pair, eof] [List → • List Pair, ( ]
[List → • Pair, eof] [List → • Pair, ( ] [Pair → • (Pair), eof]
[Pair → • (Pair), ( ] [Pair → • ( ), eof] [Pair → • ( ), ( ]
$\}$

CC Construction: goto




```
1   ALGORITHM: goto
2     INPUT: a set S of LR(1) items, a symbol X
3     OUTPUT: a set of LR(1) items
4   PROCEDURE:
5     moved := ∅
6     for item ∈ S:
7       if item = [α → β • X δ, a] then
8         moved := moved ∪ { [α → β X • δ, a] }
9       end
10    return closure(moved)
```

source subset state

target subset state

expecting to read X

X already recognized.

Analogy: ε-NFA to DFA

Subset construction (with ***lazy evaluation*** and *epsilon closures*) produces a ***DFA*** transition table

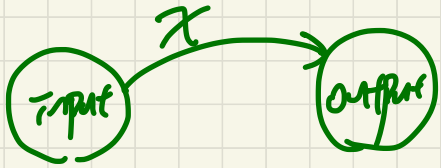
source	$d \in 0..9$	$s \in \{+,-\}$	.
$\{q_0, q_1\}$	$\{q_1, q_4\}$	$\{q_1\}$	$\{q_2\}$
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\varnothing$	$\{q_2, q_3, q_5\}$
$\{q_1\}$	$\{q_1, q_4\}$	$\varnothing$	$\{q_2\}$
$\{q_2\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$
$\{q_2, q_3, q_5\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$
$\{q_3, q_5\}$	$\{q_3, q_5\}$	$\varnothing$	$\varnothing$

target

symbol

For example, $\delta(\{q_0, q_1\}, d)$ is calculated as follows: $[d \in 0..9]$

$$\cup\{\text{ECLOSE}(q) \mid q \in \delta(q_0, d) \cup \delta(q_1, d)\}$$



ECLOSE $\left(\delta(q_0, d) \cup \delta(q_1, d) \right)$

CC Construction: goto

Calculate goto(cc_0, ( )

i.e., "next subset state" from cc_0 taking (

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

[Pair → • (Pair), ( ]
[Pair → • ( ), eof]
[Pair → • (Pair), eof]
[Pair → • ( ), ( ]

$$cc_0 = \{ [Goal \rightarrow \bullet List, \texttt{eof}]\ [List \rightarrow \bullet List\ Pair, \texttt{eof}]\ [List \rightarrow \bullet List\ Pair, (]\ [List \rightarrow \bullet Pair, \texttt{eof}]\ [List \rightarrow \bullet Pair, (]\ [Pair \rightarrow \bullet (\ Pair\ ), \texttt{eof}]\ [Pair \rightarrow \bullet (\ Pair\ ), (]\ [Pair \rightarrow \bullet (\ ), \texttt{eof}]\ [Pair \rightarrow \bullet (\ ), (] \}$$

will trigger additional items

[Pair → (• Pair), ( ]
[Pair → (•), eof]
[Pair → (• Pair), eof]
[Pair → (•), ( ]

closure

```
1  ALGORITHM: goto
2    INPUT: a set s of LR(1) items, a symbol x
3    OUTPUT: a set of LR(1) items
4  PROCEDURE:
5    moved := ∅
6    for item ∈ s:
7      if item = [α → β • x δ, a] then
8        moved := moved ∪ { [α → β x • δ, a] }
9      end
10   return closure(moved)
```

must be a terminal

Exercise: why the highlighted items trigger the two additional items

$$cc_3 = \{ [Pair \rightarrow \bullet (\ Pair\ ),)]\ [Pair \rightarrow (\bullet Pair\ ), \texttt{eof}]\ [Pair \rightarrow (\bullet Pair\ ), (]\ [Pair \rightarrow \bullet (\ ),)]\ [Pair \rightarrow (\bullet), \texttt{eof}]\ [Pair \rightarrow (\bullet), (] \}$$

CC Construction: goto

Calculate goto(cc_0, List)

i.e., "next subset state" from cc_0 taking List

1	$Goal \rightarrow List$
2	$List \rightarrow List\ Pair$
3	$\mid Pair$
4	$Pair \rightarrow (\ Pair\ )$
5	$\mid (\ )$

closure({ [Goal → List • eof], [List → List • Pair, eof], [List → List • Pair, ( ] })

$$cc_0 = \left\{ \begin{array}{lll} [Goal \rightarrow \bullet\ List,\ \texttt{eof}] & [List \rightarrow \bullet\ List\ Pair,\ \texttt{eof}] & [List \rightarrow \bullet\ List\ Pair,\ (] \\ [List \rightarrow \bullet\ Pair,\ \texttt{eof}] & [List \rightarrow \bullet\ Pair,\ (] & [Pair \rightarrow \bullet\ (\ Pair\ ),\ \texttt{eof}] \\ [Pair \rightarrow \bullet\ (\ Pair\ ),(] & [Pair \rightarrow \bullet\ (\ ),\ \texttt{eof}] & [Pair \rightarrow \bullet\ (\ ),(] \end{array} \right\}$$

Dimension 1: Two alt. for Pair

Pair → (Pair)

Pair → ( )

Dimension 2: FIRST(&a)

```
1  ALGORITHM: goto
2    INPUT: a set S of LR(1) items, a symbol X
3    OUTPUT: a set of LR(1) items
4  PROCEDURE:
5    moved := ∅
6    for item ∈ S:
7      if item = [α → β • Xδ, a] then
8        moved := moved ∪ { [α → βX • δ, a] }
9      end
10   return closure(moved)
```

↳ 1. terminal 2. variable

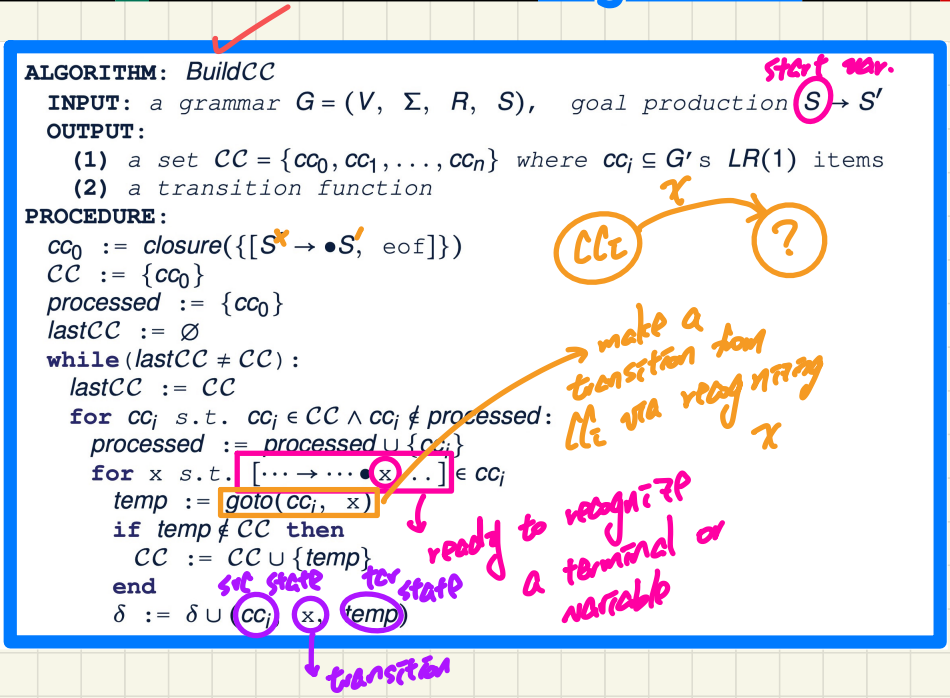
List

List

$$cc_1 = \left\{ \begin{array}{lll} [Goal \rightarrow List \bullet,\ \texttt{eof}] & [List \rightarrow List \bullet\ Pair,\ \texttt{eof}] & [List \rightarrow List \bullet\ Pair,\ (] \\ [Pair \rightarrow \bullet\ (\ Pair\ ),\ \texttt{eof}] & [Pair \rightarrow \bullet\ (\ Pair\ ),\ (] & [Pair \rightarrow \bullet\ (\ ),\ \texttt{eof}] \\ & [Pair \rightarrow \bullet\ (\ ),\ (] & \end{array} \right\}$$

CC and δ Construction: Algorithm and Exercise

```
1  ALGORITHM: BuildCC
2    INPUT: a grammar G = (V, Σ, R, S),  goal production S → S'
3    OUTPUT:
4      (1) a set CC = {cc_0, cc_1, ..., cc_n} where cc_i ⊆ G's LR(1) items
5      (2) a transition function
6  PROCEDURE:
7    cc_0 := closure({[S' → •S, eof]})
8    CC := {cc_0}
9    processed := {cc_0}
10   lastCC := ∅
11   while (lastCC ≠ CC):
12     lastCC := CC
13     for cc_i s.t. cc_i ∈ CC ∧ cc_i ∉ processed:
14       processed := processed ∪ {cc_i}
15       for x s.t. [··· → ··· •x ···] ∈ cc_i
16         temp := goto(cc_i, x)
17         if temp ∉ CC then
18           CC := CC ∪ {temp}
19         end
20         δ := δ ∪ (cc_i, x, temp)
```




```
1  Goal → List
2  List → List Pair
3       | Pair
4  Pair → ( Pair )
5       | ( )
```

Ex1. Calculate CC (i.e., all reachable subset states).

Ex2. Calculate δ (i.e., relating members of CC by terminals and non-terminals).

CC and δ Construction: Output 1

List

$$CC_0 = \left\{ \begin{array}{lll} [Goal \to \bullet\, List, \texttt{eof}] & [List \to \bullet\, List\ Pair, \texttt{eof}] & [List \to \bullet\, List\ Pair, \underline{(}] \\ [List \to \bullet\, Pair, \texttt{eof}] & [List \to \bullet\, Pair, \underline{(}] & [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \texttt{eof}] \\ [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{(}] & [Pair \to \bullet\, \underline{(}\ \underline{)}, \texttt{eof}] & [Pair \to \bullet\, \underline{(}\ \underline{)}, \underline{(}] \end{array} \right\}$$

$$CC_1 = \left\{ \begin{array}{lll} [Goal \to List \bullet, \texttt{eof}] & [List \to List \bullet\, Pair, \texttt{eof}] & [List \to List \bullet\, Pair, \underline{(}] \\ [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \texttt{eof}] & [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{(}] & [Pair \to \bullet\, \underline{(}\ \underline{)}, \texttt{eof}] \\ & [Pair \to \bullet\, \underline{(}\ \underline{)}, \underline{(}] & \end{array} \right\}$$

$$CC_2 = \left\{ [List \to Pair \bullet, \texttt{eof}] \quad [List \to Pair \bullet, \underline{(}] \right\}$$

$$CC_3 = \left\{ \begin{array}{lll} [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{)}] & [Pair \to \underline{(} \bullet\, Pair\ \underline{)}, \texttt{eof}] & [Pair \to \underline{(} \bullet\, Pair\ \underline{)}, \underline{(}] \\ [Pair \to \bullet\, \underline{(}\ \underline{)}, \underline{)}] & [Pair \to \underline{(} \bullet\, \underline{)}, \texttt{eof}] & [Pair \to \underline{(} \bullet\, \underline{)}, \underline{(}] \end{array} \right\}$$

$$CC_4 = \left\{ [List \to List\ Pair \bullet, \texttt{eof}] \quad [List \to List\ Pair \bullet, \underline{(}] \right\}$$

$$CC_5 = \left\{ [Pair \to \underline{(}\ Pair \bullet\, \underline{)}, \texttt{eof}] \quad [Pair \to \underline{(}\ Pair \bullet\, \underline{)}, \underline{(}] \right\}$$

$$CC_6 = \left\{ \begin{array}{ll} [Pair \to \bullet\, \underline{(}\ Pair\ \underline{)}, \underline{)}] & [Pair \to \underline{(} \bullet\, Pair\ \underline{)}, \underline{)}] \\ [Pair \to \bullet\, \underline{(}\ \underline{)}, \underline{)}] & [Pair \to \underline{(} \bullet\, \underline{)}, \underline{)}] \end{array} \right\}$$

$$CC_7 = \left\{ [Pair \to \underline{(}\ \underline{)} \bullet, \texttt{eof}] \quad [Pair \to \underline{(}\ \underline{)} \bullet, \underline{(}] \right\}$$

$$CC_8 = \left\{ [Pair \to \underline{(}\ Pair\ \underline{)} \bullet, \texttt{eof}] \quad [Pair \to \underline{(}\ Pair\ \underline{)} \bullet, \underline{(}] \right\}$$

$$CC_9 = \left\{ [Pair \to \underline{(}\ Pair \bullet\, \underline{)}, \underline{)}] \right\}$$

$$CC_{10} = \left\{ [Pair \to \underline{(}\ \underline{)} \bullet, \underline{)}] \right\}$$

$$CC_{11} = \left\{ [Pair \to \underline{(}\ Pair\ \underline{)} \bullet, \underline{)}] \right\}$$

CC and δ Construction: Output 2

Transition Function

Iteration	Item	Goal	List	Pair	(	)	eof
0	CC_0	Ø	CC_1	CC_2	CC_3	Ø	Ø
1	CC_1	Ø	Ø	CC_4	CC_3	Ø	Ø
	CC_2	Ø	Ø	Ø	Ø	Ø	Ø
	CC_3	Ø	Ø	CC_5	CC_6	CC_7	Ø
2	CC_4	Ø	Ø	Ø	Ø	Ø	Ø
	CC_5	Ø	Ø	Ø	Ø	CC_8	Ø
	CC_6	Ø	Ø	CC_9	CC_6	CC_{10}	Ø
	CC_7	Ø	Ø	Ø	Ø	Ø	Ø
3	CC_8	Ø	Ø	Ø	Ø	Ø	Ø
	CC_9	Ø	Ø	Ø	Ø	CC_{11}	Ø
	CC_{10}	Ø	Ø	Ø	Ø	Ø	Ø
4	CC_{11}	Ø	Ø	Ø	Ø	Ø	Ø

DFA of the LR(1) Parser

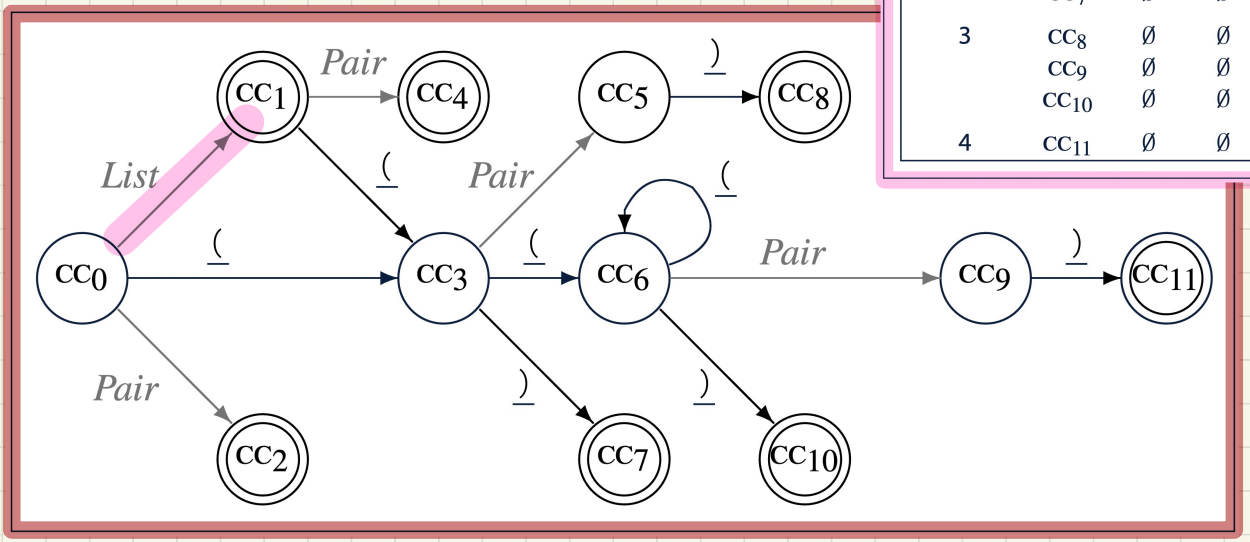


Table Construction: Algorithm

```
1   ALGORITHM: BuildActionGotoTables
2     INPUT:
3       (1) a grammar G = (V, Σ, R, S)
4       (2) goal production S → S'
5       (3) a canonical collection CC = {cc0, cc1, ..., ccn}
6       (4) a transition function δ : CC × Σ → CC
7     OUTPUT: Action Table & Goto Table
8   PROCEDURE:
9     for cci ∈ CC:
10      for item ∈ cci:
11        if item = [A → β • x γ, a] ∧ δ(cci, x) = ccj then
12          Action[i, x] := shift j
13        elseif item = [A → β •, a] then
14          Action[i, a] := reduce A → β
15        elseif item = [S → S' •, eof] then
16          Action[i, eof] := accept
17        end
18      for v ∈ V:
19        if δ(cci, v) = ccj then
20          Goto[i, v] = j
21        end
```

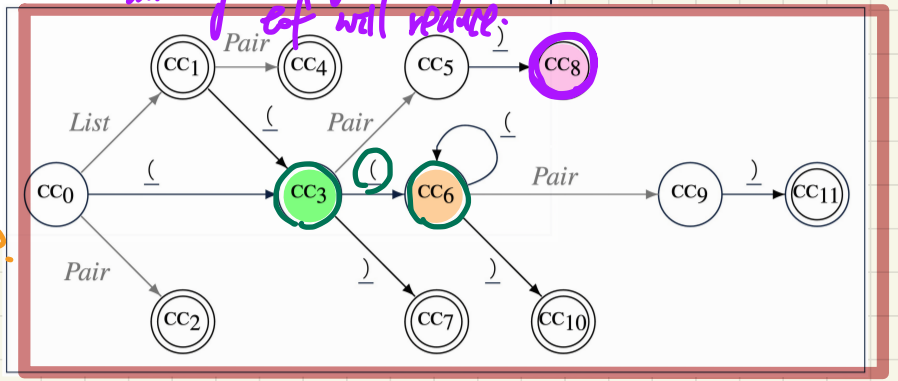
produced by BuildCC

δ(CC3, () = CC6

fill in your table.

State	Action Table			Goto Table	
	eof	(	)	List	Pair
0		s 3		1	2
1	acc	s 3			4
2	r 3	r 3			
3		s 6	s 7		5
4	r 2	r 2			
5			s 8		
6		s 6	s 10		9
7	r 5	r 5			
8	r 4	r 4			
9			s 11		
10			r 5		
11			r 4		

CC8 is about an accepting state, meaning reading eof will reduce.



$$CC_8 = \{[Pair \rightarrow (\ Pair\ )\ \bullet, \text{eof}] \quad [Pair \rightarrow (\ Pair\ )\ \bullet, (]\}$$

Bottom-Up Parsing: Discovering Ambiguities

$$CC_{13} = \left\{ \begin{array}{l} [Stmt \rightarrow \texttt{if expr then } Stmt \bullet, \{\texttt{eof}, \texttt{else}\}], \\ [Stmt \rightarrow \texttt{if expr then } Stmt \bullet \texttt{else } Stmt, \{\texttt{eof}, \texttt{else}\}] \end{array} \right\}$$

by reading eof or else, reduce to Stmt

Certain state of paser

by reading else, we shift to

What if the current word to match is else?

$\gamma\delta$ already recognized

shift or reduce to Stmt

↳ shift-reduce conflict

↳ in practice, shift will be done.

$$CC_i = \left\{ \begin{array}{l} [A \rightarrow \gamma\delta \bullet, a], \\ [B \rightarrow \gamma\delta \bullet, a] \end{array} \right\}$$

by reading a,

What if the current word to match is a?

some reduction

↳ reduce-reduce conflict

must fix the grammar.