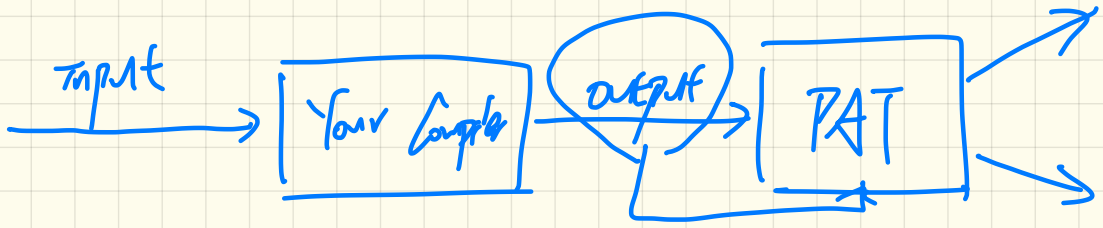
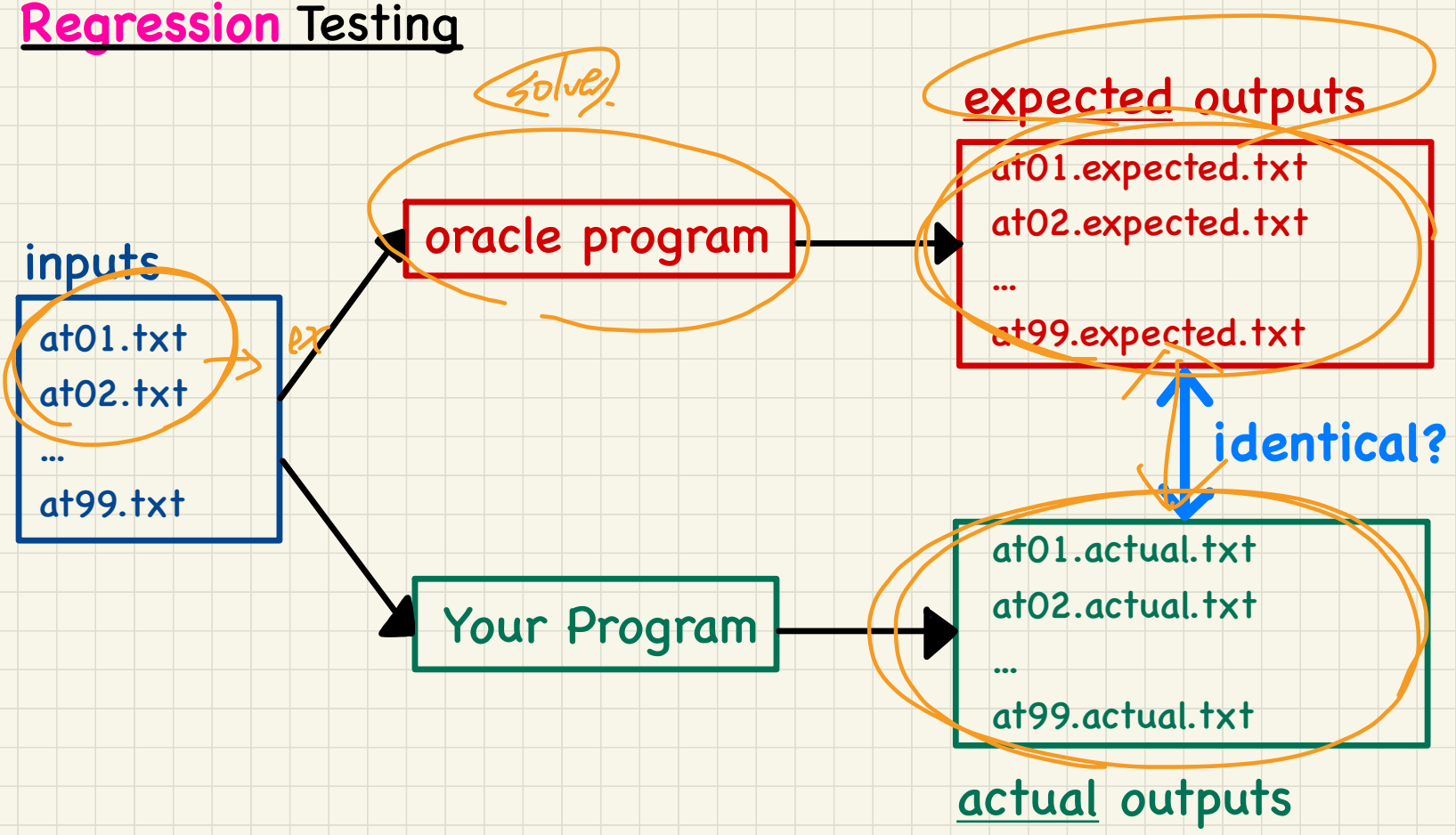
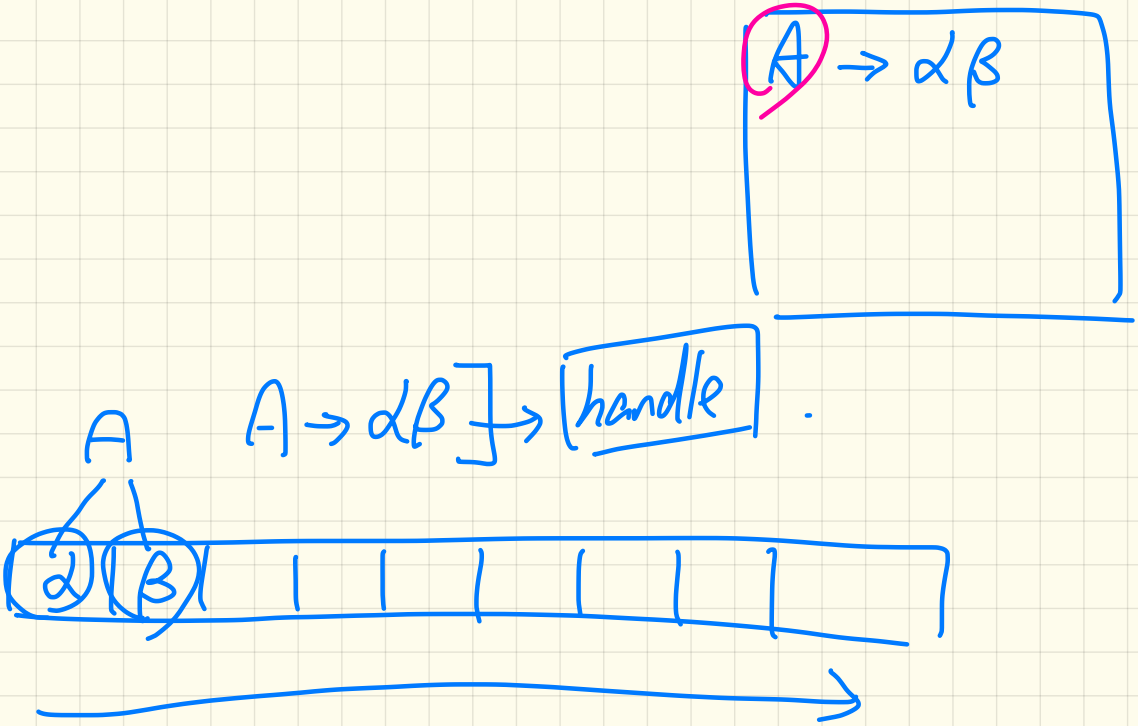
Lecture 15
Monday March 2



- Assignment 3: **Automated Regression Testing**

Regression Testing




A → αβ
A → αβ] → handle
A
α
β

Bottom-Up Parsing: Algorithm

Actions: s3 → shift to state 3
r2 → reduce to LHS of rule 2

```
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(S) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce 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 si'' then
      trace.push(word)
      trace.push(si)
      word := NextWord()
    else
      fail()
```

|cb| = 2

A → ab
trace.pop() * 4

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 (1)

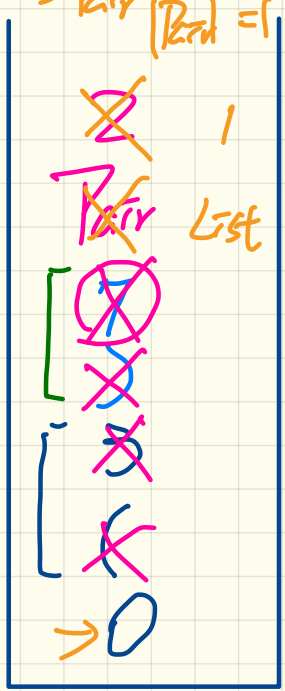
```
ALGORITHM: BUParse
  INPUT: CFG G = (V, Σ, R, S), Action & Goto Tables
  OUTPUT: Report Parse Success  or Syntax Error
PROCEDURE:
  initialize an empty stack trace
  trace.push(S) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ''accept'' then
      succeed()
    elseif act = ''reduce 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 si'' then
      trace.push(word)
      trace.push(si)
      word := NextWord()
    else
      fail()
```

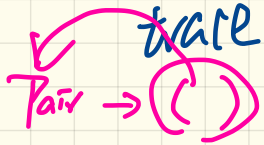
1	Goal → List
2	List → List Pair
3	| Pair
4	Pair → (Pair)
5	| ( )

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		

Parse: ( )

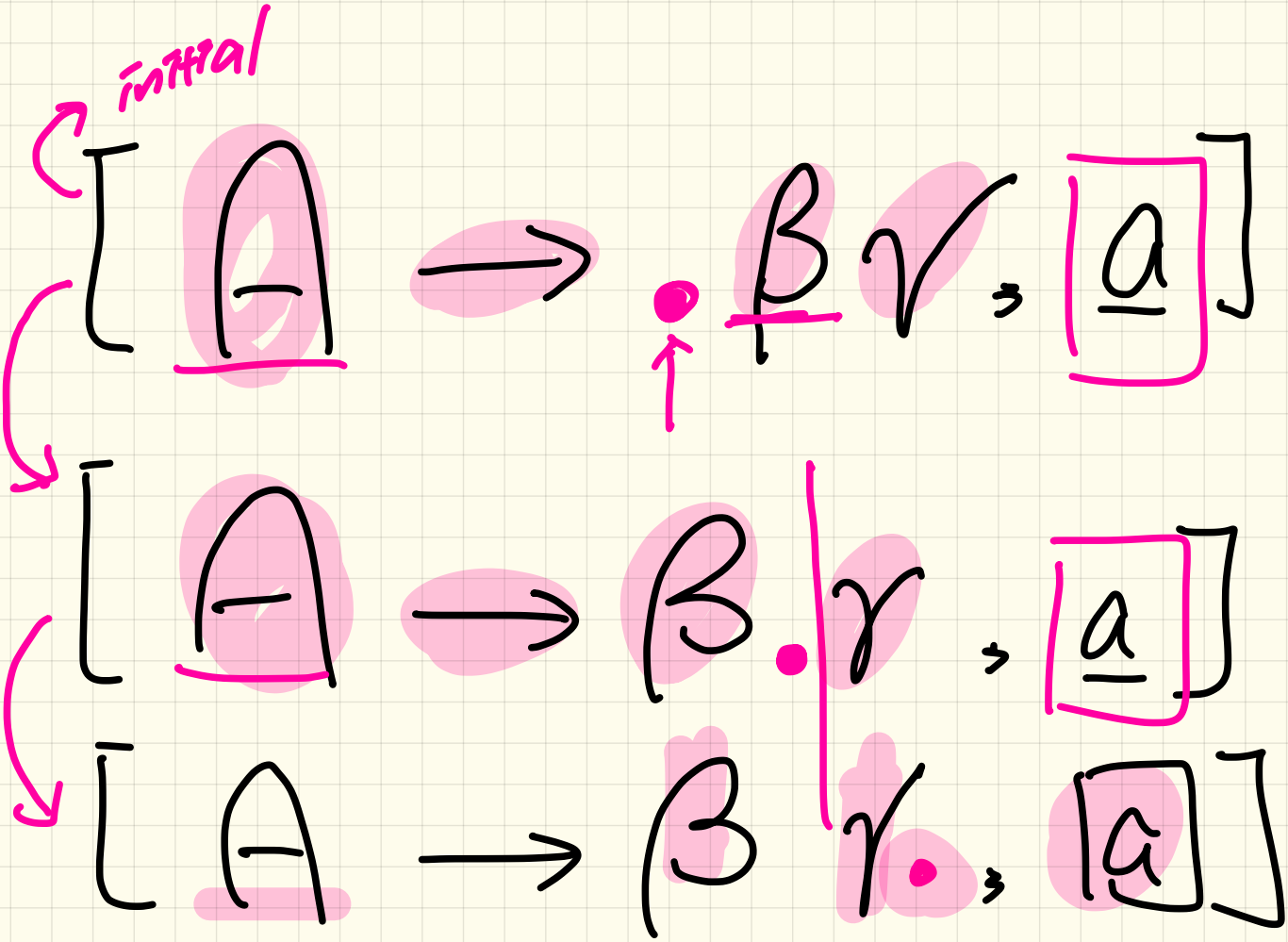
List → Pair





TD Parsing (back-track free)

BU Parsing.


initial
[A → • β γ , a]
[A → β • γ , a]
[A → β γ • , a]

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(S) /* start state */
  word := NextWord()
  while(true)
    state := trace.top()
    act := Action[state, word]
    if act = ``accept'' then
      succeed()
    elseif act = ``reduce 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 s_i'' then
      trace.push(word)
      trace.push(s_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)

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(S) /* start state */
  word := NextWord()
  while(true)
    state := trace.pop()
    act := Action[state, word]
    if act = ``accept'' then
      succeed()
    elseif act = ``reduce 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 s_i'' then
      trace.push(word)
      trace.push(s_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		