Predictive Parsing

CS 4100
Gordon Stewart
Ohio University

Quiz?

Quiz!

MCIML

- Appel 3.2: Predictive Parsing
- Reminder: Book is available online through library site; read!

Recursive Descent

Some grammars are easy to parse by hand using an algorithm you saw in 3200: ***recursive descent***

S -> if E then S else S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

Recursive Descent

Some grammars are easy to parse by hand using an algorithm you saw in 3200: ***recursive descent***

S -> if E then S else S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

Is the following string in the language?

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

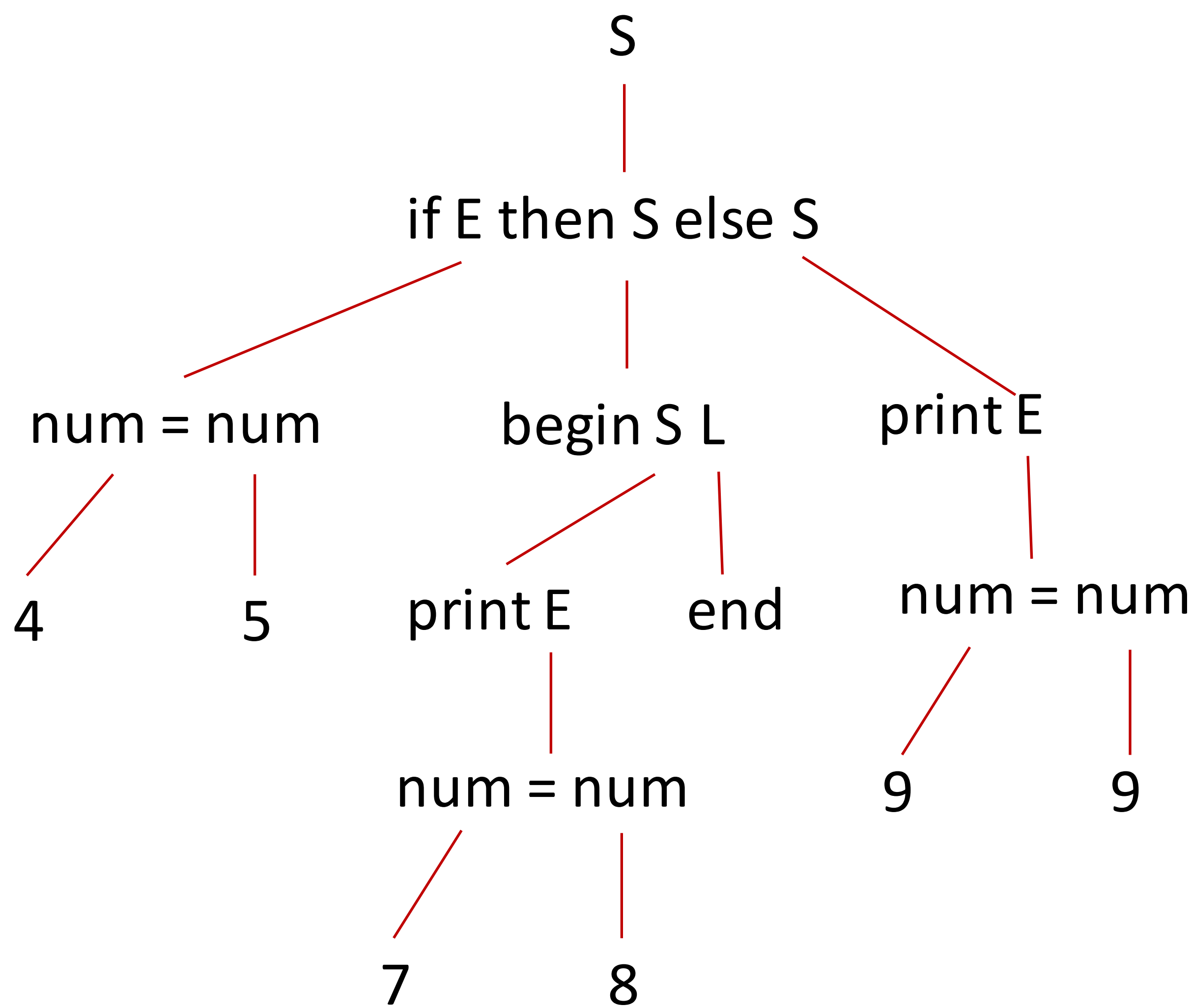
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9



Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```

S

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```

S

if E then S else S

This is the only possible rule to unfold, given first terminal in input is **if**

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```

S

if E then S else S

num = num

The next terminal in the input is a num, **4**

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

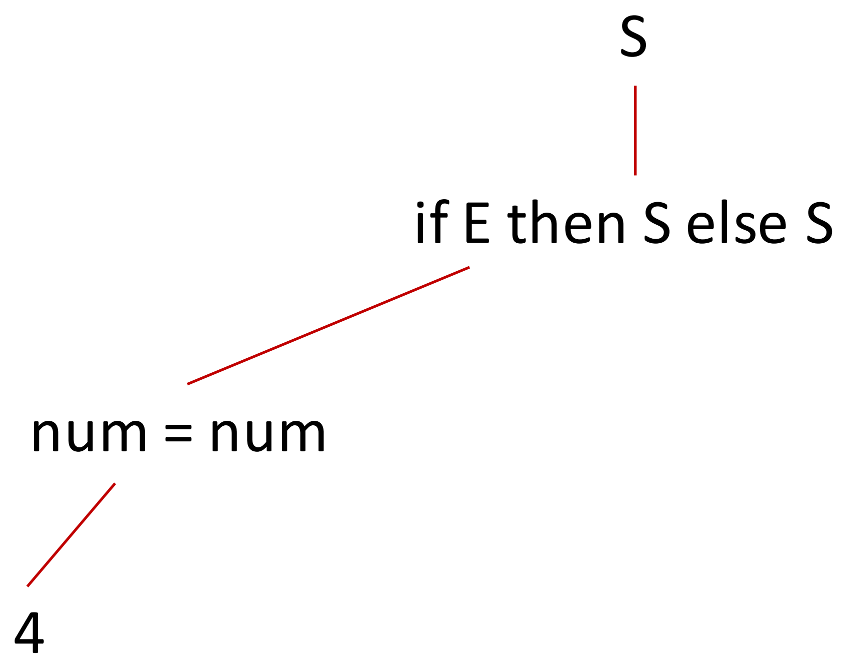
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```



Recursive Descent

S -> if E then S else S

S -> begin S L

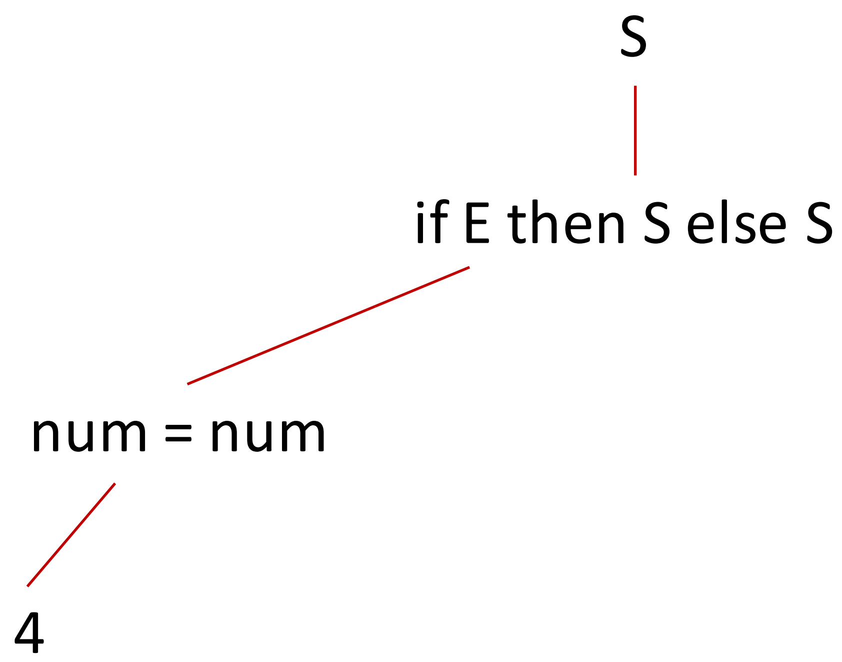
S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]



if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

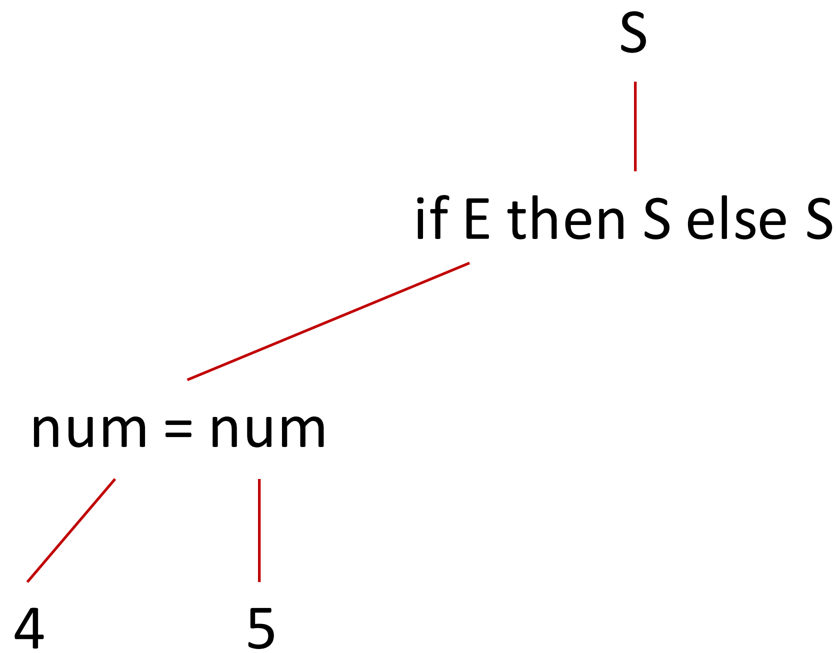
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```



Recursive Descent

S -> if E then S else S

S -> begin S L

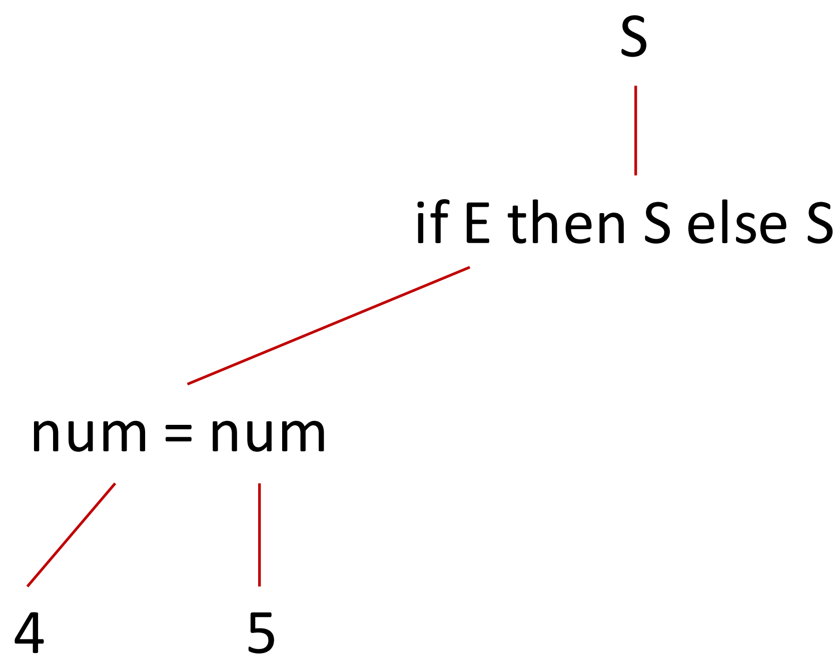
S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]




```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```

Recursive Descent

S -> if E then S else S

S -> begin S L

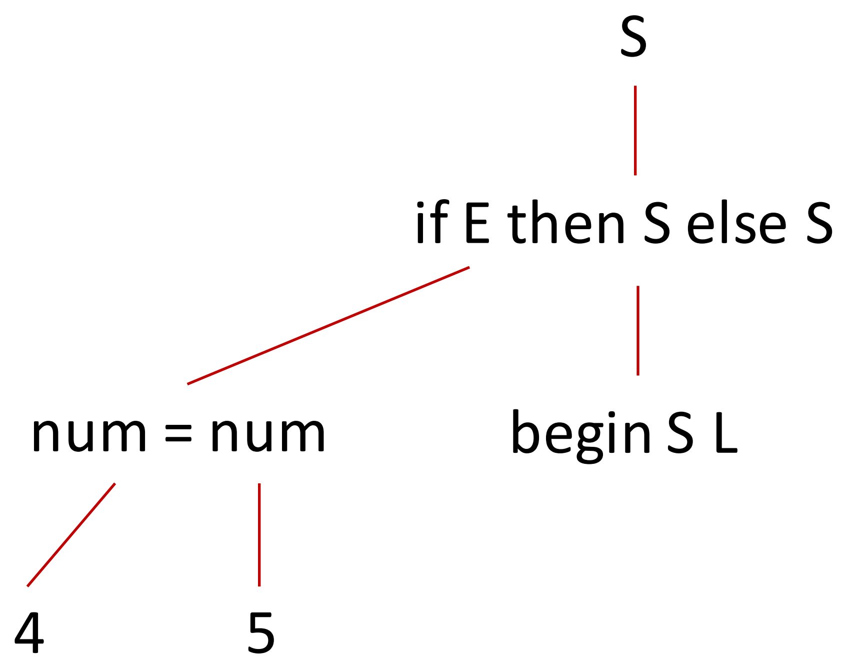
S -> print E

L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]




```
if 4 = 5 then
 begin print 7 = 8
 end
else print 9 = 9
```

Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

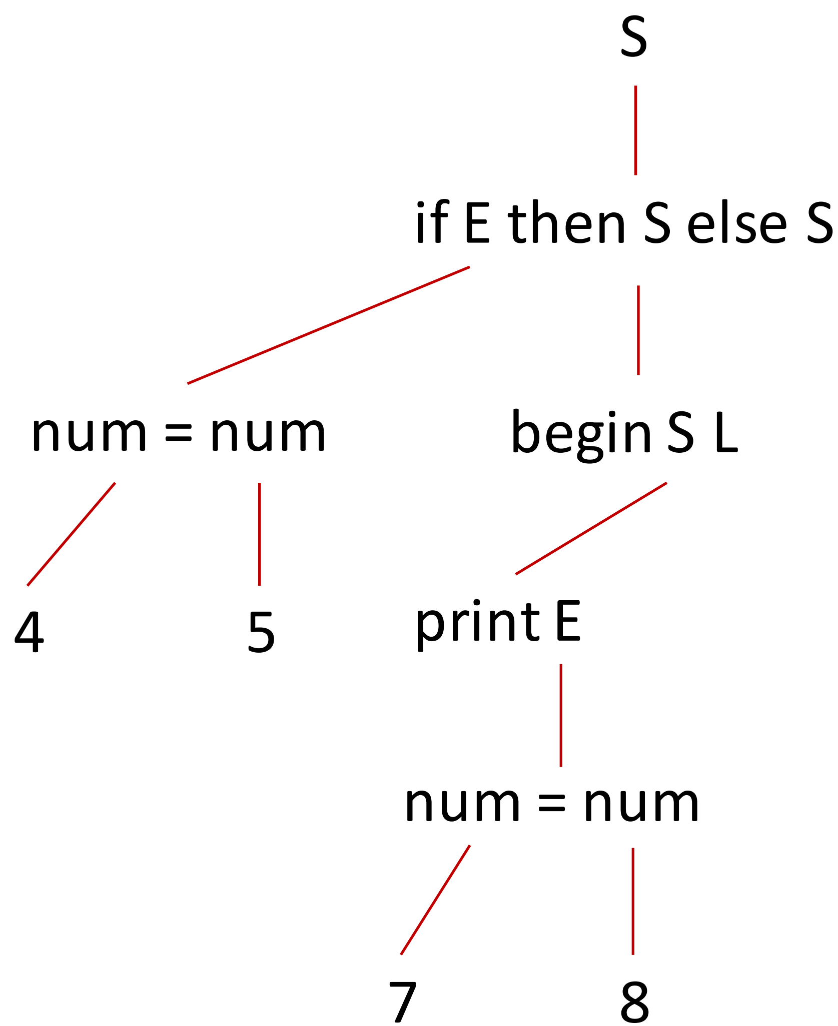
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```



Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

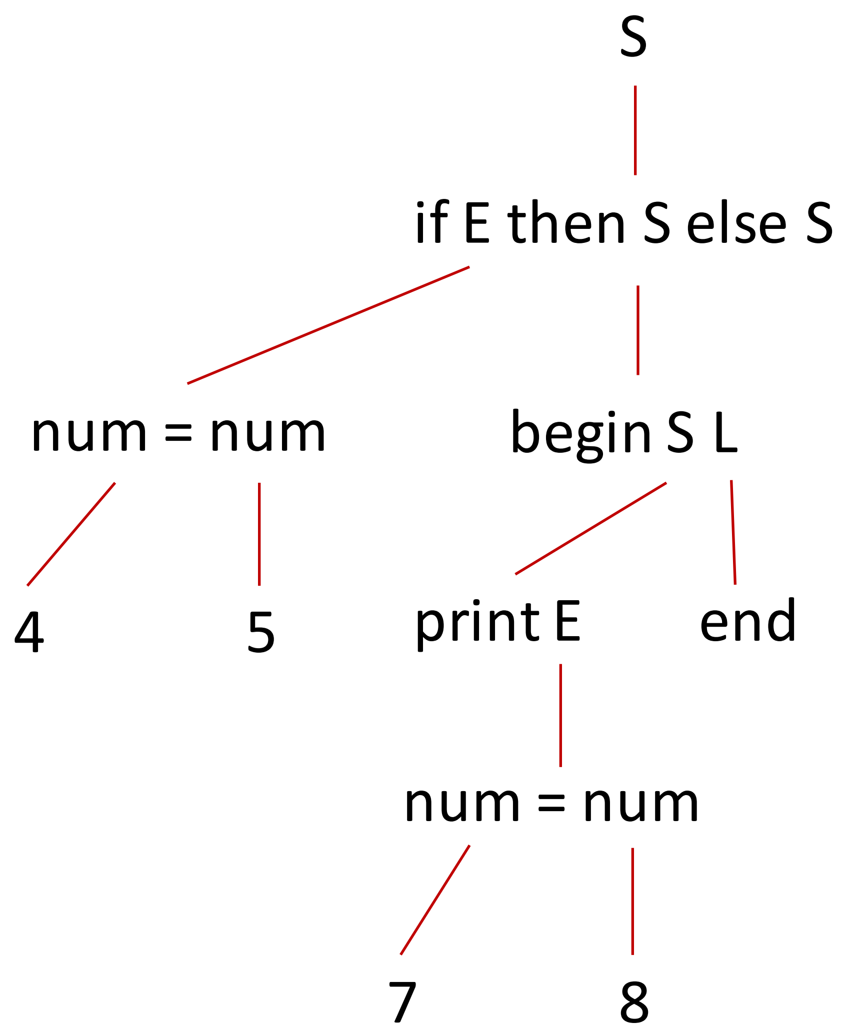
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

```
if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9
```



Recursive Descent

S -> if E then S else S

S -> begin S L

S -> print E

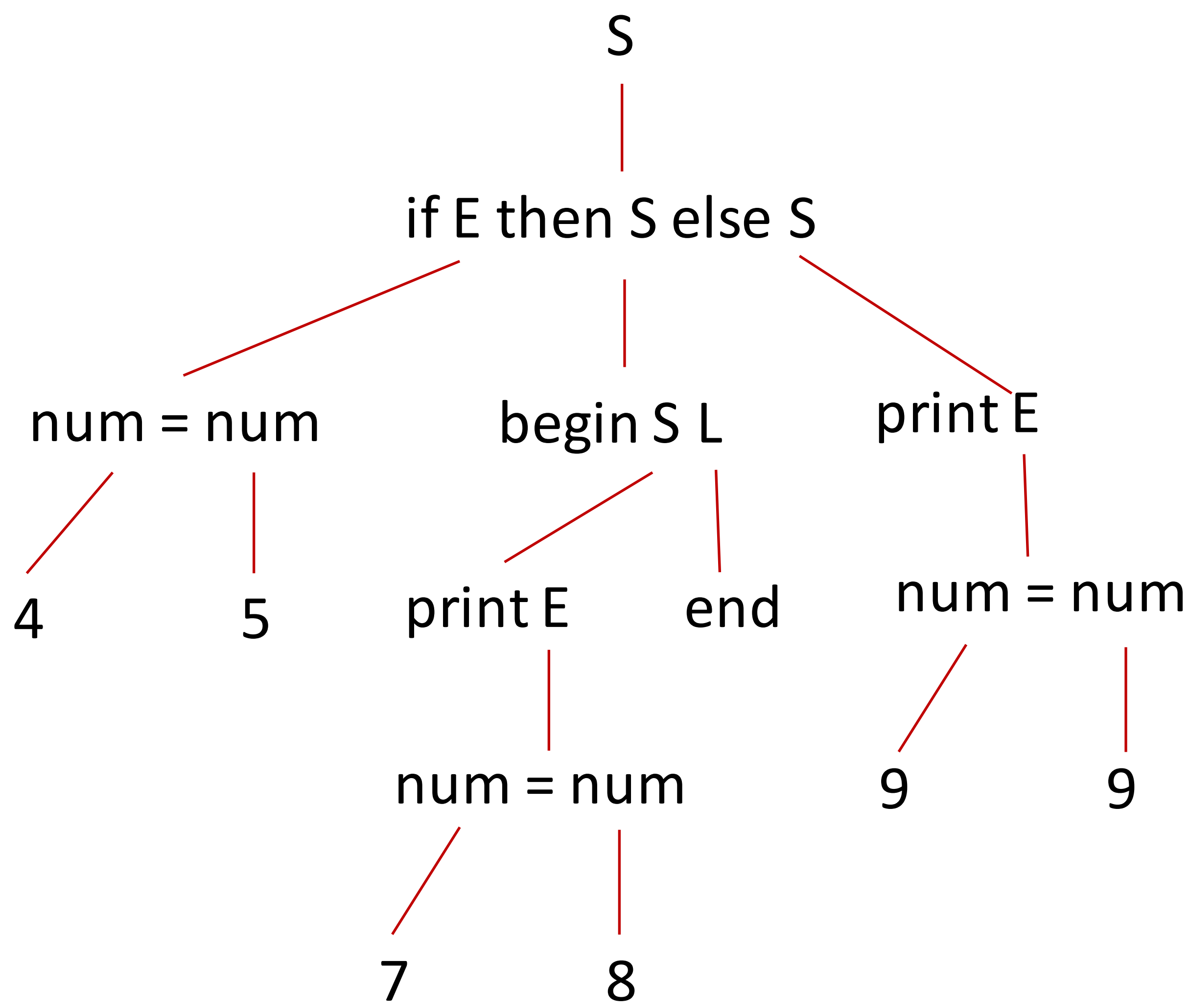
L -> end

L -> ; S L

E -> num = num

[Appel GRAMMAR 3.11]

if 4 = 5 then
  begin print 7 = 8
  end
else print 9 = 9



Recursive Descent

```
let rec s (_ : unit) : unit =
  let tok = getToken () in
  match tok with
  | IF -> eat IF; e (); eat THEN; s (); eat ELSE; s ()
  | BEGIN -> eat BEGIN; s (); l ()
  | PRINT -> eat PRINT; e ()
  | _ -> raise (Parse_failure (Some tok, "in nonterminal S"))

and l (_ : unit) : unit =
  let tok = getToken () in
  match tok with
  | END -> eat END
  | SEMI -> eat SEMI; s (); l ()
  | _ -> raise (Parse_failure (Some tok, "in nonterminal L"))

and e (_ : unit) : unit =
  let tok = getToken () in
  match tok with
  | NUM _ -> eat_num (); eat EQ; eat_num ()
  | _ -> raise (Parse_failure (Some tok, "in nonterminal E"))
```

https://github.com/gstew5/cs4100-public/blob/master/recursive/grammar311.ml

When Recursive Descent Fails

S -> if E then S else S

S -> if E then S

S -> begin S L

S -> print E

L -> end

L -> ; S L

E -> num = num

The ambiguous "dangling else" we saw last week…

This grammar can't be parsed by recursive descent;
Two productions for S have the same "first" set

Recursive Descent

```
let rec s (_ : unit) : unit =
  let tok = getToken () in
  match tok with
  | IF -> (* ??? *)
  | BEGIN -> eat BEGIN; s (); l ()
  | PRINT -> eat PRINT; e ()
  | _ -> raise (Parse_failure (Some tok, "in nonterminal S"))
```

S -> if E then S else S
S -> if E then S

Recursive descent only works when the next token in the input string gives enough information to determine which production to apply!

The grammar above is ambiguous – it derives two distinct parse trees for the string **if E then if E then S else S**.

But there are many unambiguous grammars that are still not parseable by recursive descent.

Example

- Unambiguous but not parseable by recursive descent:

```
E -> T
E -> T + E
T -> num
```

- Why?
  - Consider **3 + 4 + 5**
  - Next symbol: **3**
  - When parsing the nonterminal **E**, should we apply
    - **E -> T**, or
    - **E -> E + E**?

BUILDING RECURSIVE DESCENT PARSERS USING FIRST AND FOLLOW SETS

FIRST and FOLLOW Sets

nullable(X) = if X can derive the empty string ""

X -> Yb
Y ->
Y -> a

nullable(X) = no
nullable(Y) = yes

FIRST(γ) = the set of terminals that can begin strings derived from γ

γ is either a terminal or nonterminal

first(X) = {a, b}
first(Y) = {a}

FOLLOW(X) = the set of terminals that can immediately follow X

follow(X) = {}
follow(Y) = {b}

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \to Y_1 Y_2 \ldots Y_k$ (* in which case $X \to$ "" *)

if $Y_1 Y_2 \ldots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

```
X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY
```

	nullable
X	false
Y	false
Z	false

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$ (* in which case $X \rightarrow$ "" *)

if $Y_1 Y_2 \ldots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY

	nullable
X	false
Y	false
Z	false

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \to Y_1 Y_2 \ldots Y_k$ (* in which case $X \to$ “” *)

if $Y_1 Y_2 \ldots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

```
X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY
```

	nullable
X	false
Y	**true**
Z	false

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$ (* in which case $X \rightarrow$ "" *)

if $Y_1 Y_2 \ldots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

```
X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY
```

	nullable
X	**true**
Y	true
Z	false

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \to Y_1 Y_2 \dots Y_k$ (* in which case $X \to$ "" *)

if $Y_1 Y_2 \dots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

```
X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY
```

	nullable
X	true
Y	true
Z	**true**

Iterative Calculation of **nullable**

initialize a table **nullable** to all **false**.

repeat

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$ (* in which case $X \rightarrow$ "" *)

if $Y_1 Y_2 \ldots Y_k$ are all nullable, or k = 0

nullable(X) = **true**

until **nullable** table did not change in this iteration.

X -> Yb
X -> Y
Y ->
Y -> Xa
Z -> cX
Z -> XY

	nullable
X	true
Y	true
Z	true

Nothing changed in this iteration so we're done
(we've reached a "fixed point" of the iterative analysis)

Iterative Solution of Recursive Equations

nullable calculation specific instance of a general algorithm for iteratively calculating solutions of recursive equations:

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))
nullable(Y) \/= true
nullable(Y) \/= nullable(X)
nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

ALGORITHM:

1. Construct a table setting nullable(S) = false, for each symbol S.
2. Iteratively update table using equations above until it doesn't change any more (you've reached a "fixed point" of the set of equations).

Iterative Solution of Recursive Equations

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	false
Y	false
Z	false

Iterative Solution of Recursive Equations

nullable(X) $\bigvee$= nullable(Y) $\bigvee$ (nullable(X) $\bigwedge$ nullable(Y))

nullable(Y) $\bigvee$= true

nullable(Y) $\bigvee$= nullable(X)

nullable(Z) $\bigvee$= nullable(X) $\bigwedge$ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	false
Y	**true**
Z	false

Iterative Solution of Recursive Equations

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	false
Y	true
Z	false

Iterative Solution of Recursive Equations

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	false
Y	true
Z	false

Iterative Solution of Recursive Equations

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	**true**
Y	true
Z	false

ITERATION 2

Iterative Solution of Recursive Equations

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

nullable(X) $\bigvee$= nullable(Y) $\bigvee$ (nullable(X) $\bigwedge$ nullable(Y))

nullable(Y) $\bigvee$= true

nullable(Y) $\bigvee$= nullable(X)

nullable(Z) $\bigvee$= nullable(X) $\bigwedge$ nullable(Y)

	nullable
X	true
Y	true
Z	false

ITERATION 2

Iterative Solution of Recursive Equations

$\text{nullable}(X) \bigvee= \text{nullable}(Y) \bigvee (\text{nullable}(X) \bigwedge \text{nullable}(Y))$

$\text{nullable}(Y) \bigvee= \text{true}$

$\boldsymbol{\text{nullable}(Y) \bigvee= \text{nullable}(X)}$

$\text{nullable}(Z) \bigvee= \text{nullable}(X) \bigwedge \text{nullable}(Y)$

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	true
Y	true
Z	false

ITERATION 2

Iterative Solution of Recursive Equations

nullable(X) $\bigvee$= nullable(Y) $\bigvee$ (nullable(X) $\bigwedge$ nullable(Y))

nullable(Y) $\bigvee$= true

nullable(Y) $\bigvee$= nullable(X)

nullable(Z) $\bigvee$= nullable(X) $\bigwedge$ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	true
Y	true
Z	**true**

ITERATION 2

Iterative Solution of Recursive Equations

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

ITERATION 3

	nullable
X	true
Y	true
Z	true

Iterative Solution of Recursive Equations

$\text{nullable}(X) \bigvee= \text{nullable}(Y) \bigvee (\text{nullable}(X) \bigwedge \text{nullable}(Y))$

$\text{nullable}(Y) \bigvee= \text{true}$

$\text{nullable}(Y) \bigvee= \text{nullable}(X)$

$\text{nullable}(Z) \bigvee= \text{nullable}(X) \bigwedge \text{nullable}(Y)$

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

ITERATION 3

	nullable
X	true
Y	true
Z	true

Iterative Solution of Recursive Equations

$\text{nullable}(X) \bigvee= \text{nullable}(Y) \bigvee (\text{nullable}(X) \bigwedge \text{nullable}(Y))$

$\text{nullable}(Y) \bigvee= \text{true}$

$\text{nullable}(Y) \bigvee= \text{nullable}(X)$

$\text{nullable}(Z) \bigvee= \text{nullable}(X) \bigwedge \text{nullable}(Y)$

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	true
Y	true
Z	true

ITERATION 3

Iterative Solution of Recursive Equations

nullable(X) \/= nullable(Y) \/ (nullable(X) /\ nullable(Y))

nullable(Y) \/= true

nullable(Y) \/= nullable(X)

nullable(Z) \/= nullable(X) /\ nullable(Y)

X -> Y
X -> XY
Y ->
Y -> X
Z -> XY

	nullable
X	true
Y	true
Z	true

Nothing changed in this iteration so we're done.

ITERATIVE CALCULATION OF FIRST AND FOLLOW SETS

FIRST and FOLLOW Sets

nullable(X) = if X can derive the empty string ""

X -> Yb
Y ->
Y -> a

nullable(X) = yes
nullable(Y) = yes

FIRST(γ) = the set of terminals that can begin strings derived from γ

γ is either a terminal or nonterminal

first(X) = {a, b}
first(Y) = {a}

FOLLOW(X) = the set of terminals that can immediately follow X

follow(X) = {}
follow(Y) = {b}

First and Follow Equations

initialize tables *FIRST* and *FOLLOW* to empty.

for each terminal symbol **a**, FIRST(**a**) = {**a**}

repeat

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$$

until *FIRST* and *FOLLOW* did not change in this iteration.

First and Follow Equations

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

X -> $\underbrace{\text{A B C D E F G}}_{\text{all nullable}}$ H I

FIRST(X) = FIRST(X) ∪ FIRST(H)

First and Follow Equations

if $Y_1 \dots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

$X \rightarrow \underbrace{A\ B\ C\ D\ E\ F\ G}_{\text{all nullable}}\ H\ I$

FIRST(X) = FIRST(X) ∪ FIRST(H)

if $Y_{i+1} \dots Y_k$ are all nullable, or if i = k

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$

$X \rightarrow A\ B\ C\ D\ E\ F\ G\ H\ \underbrace{I}_{\text{nullable}}$

FOLLOW(H) = FOLLOW(H) ∪ FOLLOW(X)

First and Follow Equations

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

X -> $\underbrace{\text{A B C D E F G}}_{\text{all nullable}}$ H I

FIRST(X) = FIRST(X) ∪ FIRST(H)

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$

X -> A B C D E F G H $\underbrace{\text{I}}_{\text{nullable}}$

FOLLOW(H) = FOLLOW(H) ∪ FOLLOW(X)

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$$

X -> A $\underbrace{\text{B C D E F}}_{\text{all nullable}}$ G H I

FOLLOW(A) = FOLLOW(A) ∪ FIRST(G)

First and Follow Example

	nullable
X	true
Y	true
Z	false

for each terminal symbol **a**, FIRST(**a**) = {**a**}

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$$

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{}	{}
Y	{}	{}
Z	{}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

for each terminal symbol **a**, FIRST(**a**) = {**a**}

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a}	{}
Y	{}	{}
Z	{}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

for each terminal symbol **a**, FIRST(**a**) = {**a**}
for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$
for each i from 1 to k, for each j from i+1 to k
if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1
$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$
if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$
if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

	FIRST	FOLLOW
X	{a}	{}
Y	{}	{}
Z	{}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

for each terminal symbol **a**, FIRST(**a**) = {**a**}
for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$
 for each i from 1 to k, for each j from i+1 to k
  if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1
   $FIRST(X) = FIRST(X) \cup FIRST(Y_i)$
  if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k
   $FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$
  if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j
   $FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

	FIRST	FOLLOW
X	{a}	{}
Y	{c}	{}
Z	{}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

for each terminal symbol **a**, FIRST(**a**) = {**a**}

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a}	{}
Y	{c}	{}
Z	{d}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

for each terminal symbol **a**, FIRST(**a**) = {**a**}

for each production $X \rightarrow Y_1 Y_2 \dots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \dots Y_{i-1}$ are all nullable, or if i = 1

$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$

if $Y_{i+1} \dots Y_k$ are all nullable, or if i = k

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$

if $Y_{i+1} \dots Y_{j-1}$ are all nullable, or if i+1 = j

$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

	nullable
X	true
Y	true
Z	false

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

for each terminal symbol **a**, FIRST(**a**) = {**a**}
for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$
for each i from 1 to k, for each j from i+1 to k
if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1
$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$
if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k
$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$
if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j
$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$$

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

for each terminal symbol **a**, FIRST(**a**) = {**a**}

for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$

for each i from 1 to k, for each j from i+1 to k

if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1

$$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$$

if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$$

if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j

$$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$$

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

	nullable
X	true
Y	true
Z	false

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

for each terminal symbol **a**, FIRST(**a**) = {**a**}
for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$
for each i from 1 to k, for each j from i+1 to k
if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1
$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$
if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$
if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}
a	{a}	
c	{c}	
d	{d}	

First and Follow Example

for each terminal symbol **a**, FIRST(**a**) = {**a**}
for each production $X \rightarrow Y_1 Y_2 \ldots Y_k$
for each i from 1 to k, for each j from i+1 to k
if $Y_1 \ldots Y_{i-1}$ are all nullable, or if i = 1
$FIRST(X) = FIRST(X) \cup FIRST(Y_i)$
if $Y_{i+1} \ldots Y_k$ are all nullable, or if i = k
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FOLLOW(X)$
if $Y_{i+1} \ldots Y_{j-1}$ are all nullable, or if i+1 = j
$FOLLOW(Y_i) = FOLLOW(Y_i) \cup FIRST(Y_j)$

	nullable
X	true
Y	true
Z	false

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}
a	{a}	
c		
d		

Nothing changed in this iteration so we're done
(we've reached a "fixed point" of the iterative analysis)

FIRST Equations Applied

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

FIRST(X) U= FIRST(Y) U {a}

FIRST(Y) U= {c}

FIRST(Z) U= {d} U FIRST(X) U FIRST(Y)

FOLLOW(X) U= FIRST(Y) U FIRST(Z)

FOLLOW(Y) U= FIRST(Z)

FOLLOW(Z) U= {}

BUILDING PREDICTIVE PARSING TABLES FROM FIRST AND FOLLOW

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X			
Y			
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z … W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z … W are **all nullable**, add production X -> Y Z … W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	c	d
X			
Y			
Z			

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y		
Y			
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z … W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z … W are **all nullable**, add production X -> Y Z … W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y	X - > Y	
Y			
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z … W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z … W are **all nullable**, add production X -> Y Z … W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y	X - > Y	X -> Y
Y			
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y	X -> Y	X -> Y
Y	Y ->		
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	
Z			

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	c	d
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	Y ->
Z			

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z … W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z … W are **all nullable**, add production X -> Y Z … W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	**c**	**d**
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	Y ->
Z	Z -> X Y Z		

	FIRST	**FOLLOW**
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z … W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z … W are **all nullable**, add production X -> Y Z … W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	c	d
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	Y ->
Z	Z -> X Y Z	Z -> X Y Z	

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

To build table ***M***:

- In ***M***[X, a], add all productions X -> Y Z ... W such that **a** is in FIRST(Y) or nullable(Y) and **a** is in FIRST(Z), etc., or
- If Y Z ... W are **all nullable**, add production X -> Y Z ... W to ***M***[X, a] if **a** in FOLLOW(X).

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	c	d
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	Y ->
Z	Z -> X Y Z	Z -> X Y Z	Z -> d Z -> X Y Z

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Predictive Parsing Tables

Duplicate entries in the parsing table mean this grammar **can't be parsed** by recursive descent.

X -> Y
X -> a
Y ->
Y -> c
Z -> d
Z -> X Y Z

	a	c	d
X	X -> a X -> Y	X - > Y	X -> Y
Y	Y ->	Y -> c Y ->	Y ->
Z	Z -> X Y Z	Z -> X Y Z	Z -> d Z -> X Y Z

	FIRST	FOLLOW
X	{a, c}	{a, c, d}
Y	{c}	{a, c, d}
Z	{a, c, d}	{}

	nullable
X	true
Y	true
Z	false

Left-Recursive Grammars

Consider the following grammar:

```
E -> T
E -> E + E
T -> num
```

This grammar is not parseable via recursive descent. Why?

Left-Recursive Grammars

Consider the following grammar:

E -> T
E -> E + E
T -> num

Grammars with ***left recursion*** are not parseable via recursive descent

This grammar is not parseable via recursive descent. Why?

	FIRST	FOLLOW
E	**at least** FIRST(T) (i.e., num)	...
T	num	...

	num
E	E -> T E -> E + E
T	T -> num

Factoring Left-Recursive Grammars

To eliminate left recursion, rewrite using ***right recursion***:

E -> T
E -> E + E
T -> num

refactor

E -> T E' $

E' -> + T E'
E' ->

T -> num

	FIRST	FOLLOW
E	num	
E'	+	$
T	num	+, $

	num	+	$
E	E -> T E'		
E'		E' -> + T E'	E' ->
T	T -> num		

Left Factoring

If two productions for the same nonterminal start with the same symbol, we can ***left factor*** the grammar.

S -> if E then S else S
S -> if E then S
S -> …

refactor

S' -> S $
S -> if E then S X
S -> …
X ->
X -> else S

	FIRST	FOLLOW
S'	if	
S	if	$, else
X	else	$
E	…	then

	if	then	else	$
S'	S' -> S $			
S	S -> if E then S X			
X			X -> else S	X ->