PROJECT 2: Useless Symbols, FIRST and FOLLOW sets, and Predictive Parsing

CSE 340 Spring 2017

Rida A. Bazzi

Project 2 Goals

- I have introduced in class predictive parsing and FIRST and FOLLOW sets
- The goal of this project is to show you how the process of building a predictive parser can be automated
- Another important goal of the project is to give you experience in writing a substantial program which is non-trivial conceptually
  - This will make you a better programmer
  - You will have a better understanding of the power of abstraction in building code
  - You will have a better appreciation of the material covered so far

Outline

- Set representation
- Grammar representation
- Calculating useless symbols
- Calculating FIRST sets
- Calculating FOLLOW sets
- Determining if a grammar has a predictive parser

Bit-vector representation

- We can assign an index to each element in our set. Here we have:

“a” has index 0

“b” has index 1

“c” has index 2

…

0	1	2	3	4
“a”	“b”	“c”	“d”	“e”

Bit-vector representation

- We can assign an index to each element in our set. Here we have:

“a” has index 0

“b” has index 1

“c” has index 2

…

0	1	2	3	4
“a”	“b”	“c”	“d”	“e”

- A set is a bit vector (or Boolean vector) indicating which elements are in the set and which elements are not in the set

Bit-vector representation

- We can assign an index to each element in our set. Here we have:

0	1	2	3	4
“a”	“b”	“c”	“d”	“e”

- A set is a bit vector (or Boolean vector) indicating which elements are in the set and which elements are not in the set

EXAMPLES

	0	1	2	3	4
{ “a”, “b” }	T	T	F	F	F

{ “a”, “b”, “c”, “d”, “e” }

Bit-vector representation

- We can assign an index to each element in our set. Here we have:

0	1	2	3	4
"a"	"b"	"c"	"d"	"e"

- A set is a bit vector (or Boolean vector) indicating which elements are in the set and which elements are not in the set

EXAMPLES

	0	1	2	3	4
{ "a", "b" }	T	T	F	F	F
	{ "a",	"b",	~~"c"~~,	~~"d"~~,	~~"e"~~ }
	✓	✓			

Bit-vector representation

- We can assign an index to each element in our set. Here we have:

0	1	2	3	4
"a"	"b"	"c"	"d"	"e"

- A set is a bit vector (or Boolean vector) indicating which elements are in the set and which elements are not in the set

EXAMPLES

{ "a", "b" , "e"}

0	1	2	3	4
T	T	F	F	T

{ "a",	"b",	"~~c~~",	"~~d~~",	"e" }
✓	✓			✓

Operations on sets: Union

Example: { "a", "c" } ∪ { "a", "e" }

	0	1	2	3	4
{ "a", "c" }	T	T	T	F	F

	0	1	2	3	4
{ "a", "e" }	T	T	F	F	T

	0	1	2	3	4
{ "a", "c", "e" }	T	T	T	F	T

In general

S1, S2, S3

```
for i = 0 to universe_size - 1
    S3[i] = S1[i] or S2[i]
```

Operations on sets: Membership

```
boolean  is_element(S : set, i : integer)
{
          if ( (i > = 0) && (i < universe_size – 1)  )
                    return S[i];
          else
                    return false;
}
```

Operations on sets: Printing a set

	0	1	2	3	4
universe	“a”	“b”	“c”	“d”	“e”

```
boolean  print_set(S : set)
{
        for ( i = 0 to universe_size – 1 )
            if S[i] then
                print universe[i];          // this does not print commas
                                            // or parentheses
}
```

Summary

- Universe array contains the actual names of the elements
- For all set manipulations an element is simply an index
- An element is in a set if the array entry corresponding to the element (index) is true
- To print an element, print the corresponding entry in the universe array

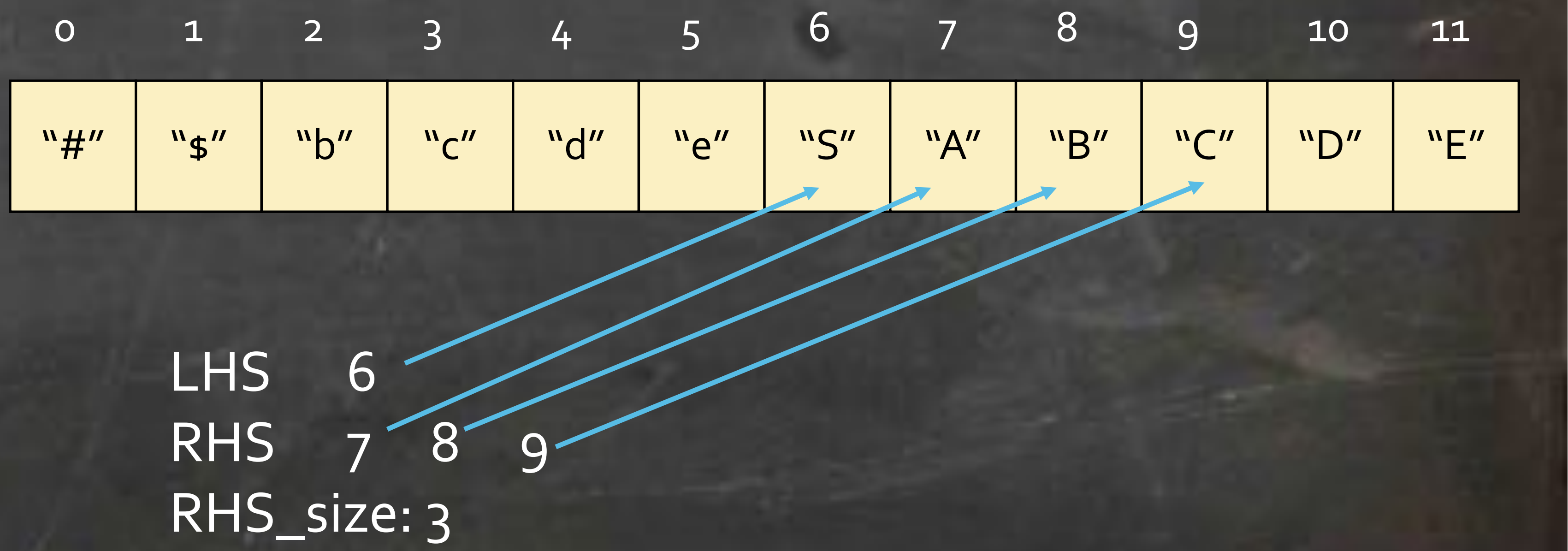
Grammar Representation

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Universe for FIRST and FOLLOW sets

0	1	2	3	4	5
"#"	"$"	"b"	"c"	"d"	"e"

All Symbols



0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

Rule 1

LHS 6
RHS 7 8 9
RHS_size: 3

Grammar Representation

S → A B C (1)

A → D E (2)

B → b B (3)

B → ε (4)

C → c C (5)

C → ε (6)

D → d D (7)

D → ε (8)

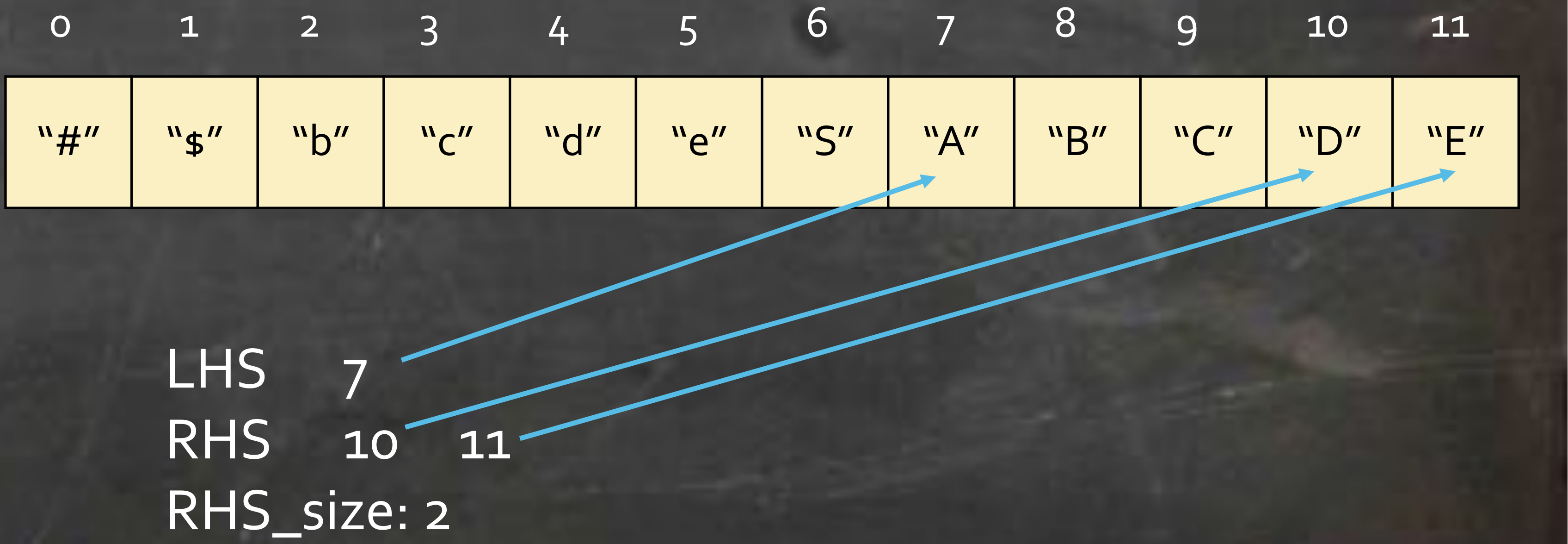
E → e E (9)

E → ε (10)

Universe for FIRST and FOLLOW sets

0	1	2	3	4	5
"#"	"$"	"b"	"c"	"d"	"e"

All Symbols



0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

Rule 2

LHS 7

RHS 10 11

RHS_size: 2

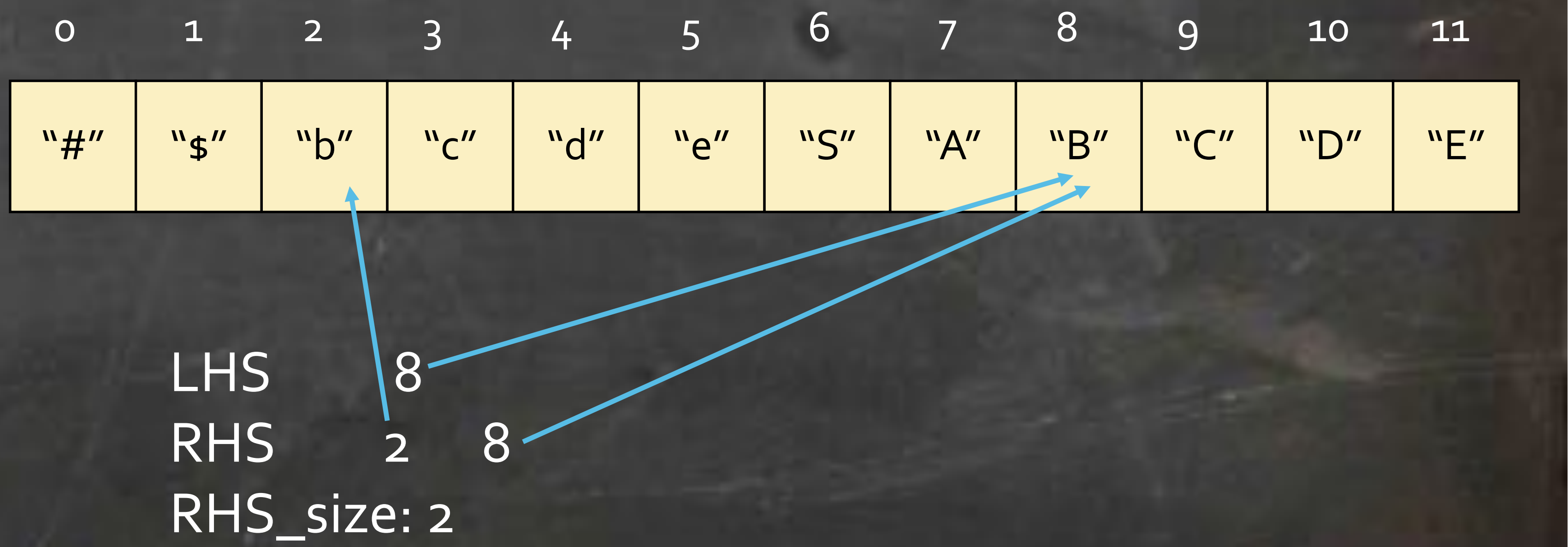
Grammar Representation

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Universe for FIRST and FOLLOW sets

0	1	2	3	4	5
"#"	"$"	"b"	"c"	"d"	"e"

All Symbols



0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

Rule 2

LHS 8
RHS 2 8
RHS_size: 2

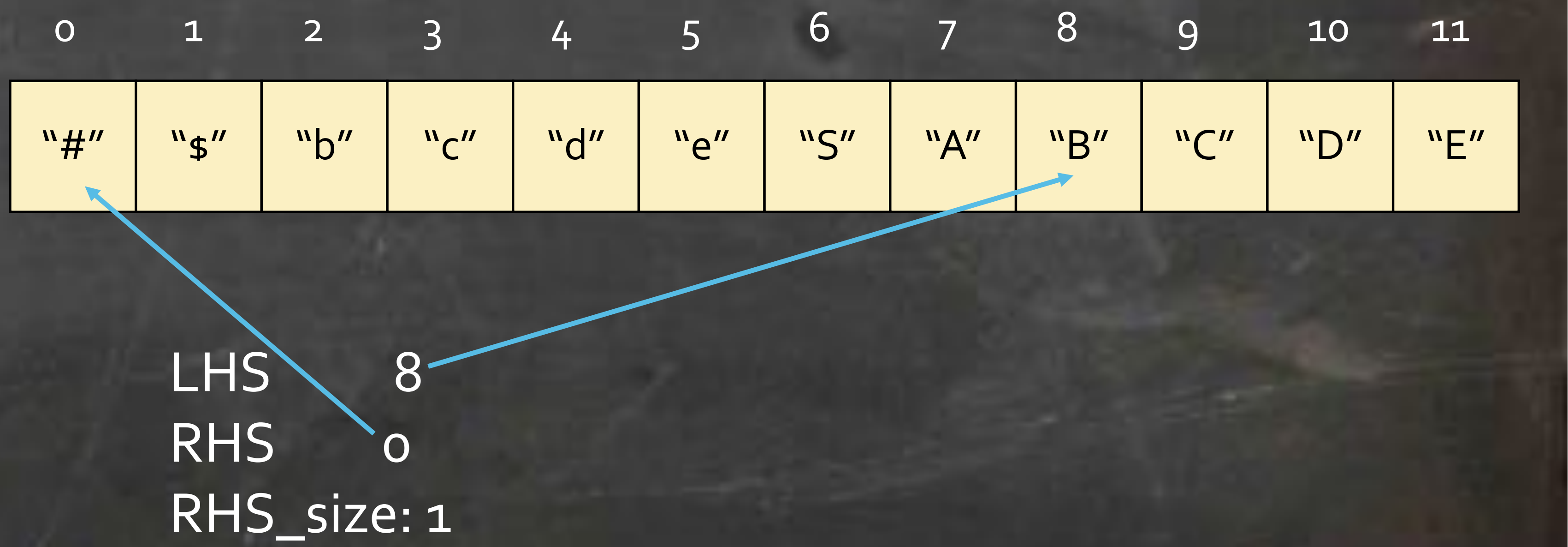
Grammar Representation

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Universe for FIRST and FOLLOW sets

0	1	2	3	4	5
“#”	“$”	“b”	“c”	“d”	“e”

All Symbols



0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

Rule 2

LHS 8
RHS 0
RHS_size: 1

Useless Symbols

A symbol is useless if it does not appear in the derivation of a string of terminals or in the derivation of the empty string

A symbol is not useless if it appears in the derivation of a string of terminals or in a derivation of the empty string

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w \in T^*$$

Calculating Useless Symbols

- We start by calculating generating symbols
  - A symbol is generating if it can derive a string in T* (zero or more sequence of terminals)
- Then we determine reachable symbols
  - A symbol A is reachable if S can derive a sentential form containing the symbol:

$$S \overset{*}{\Rightarrow} x\,A\,y$$

Calculating generating symbols

1. Initialization
   - all terminals are generating
   - ε is generating

2. If $A \rightarrow A_1\, A_2 \ldots A_k$ is a grammar rule and
   - A_1 generating and
   - A_2 generating and
   - … and
   - …
   - A_k generating

A is generating

Iterative approach to calculating generating symbols

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

Generating array

0	F
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Initialization

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- No change

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- No change

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- No change

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- B is generating

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- B's entry changes to true

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- No change

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C	(1)
A → D E	(2)
B → b B	(3)
B → ε	(4)
C → c C	(5)
C → ε	(6)
D → d D	(7)
D → ε	(8)
E → e E	(9)
E → ε	(10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- C is generating

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	F
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- C's entry changes to true

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	T
10	F
11	F

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- At the end of the first round (going over all rules), we get the array on the right
- Since some entries have changed, we need to do another round

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- At the end of the first round (going over all rules), we get the array on the right
- Since some entries have changed, we need to do another round

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- We examine the first rule again, but we cannot tell that S is generating because A is not generating

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- We examine the second rule and now we can tell that A is generating

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	F
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)

A → D E (2)

B → b B (3)

B → ε (4)

C → c C (5)

C → ε (6)

D → d D (7)

D → ε (8)

E → e E (9)

E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- So we change A’s entry to true

Generating array

0	T	
1	F	
2	T	
3	T	
4	T	
5	T	
6	F	
7	T	←
8	T	
9	T	
10	T	←
11	T	←

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- The remaining rules do not result in any change
- But since some entries have changed in the second round, we need to do a third round

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	F
7	T
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

- In the third round, we determine that S is generating
- Since some entries changed in the third round, we need to do a fourth round

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T

Iterative approach to calculating generating symbols: Iteration

S → A B C (1)
A → D E (2)
B → b B (3)
B → ε (4)
C → c C (5)
C → ε (6)
D → d D (7)
D → ε (8)
E → e E (9)
E → ε (10)

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

- In the fourth round nothing changes and we have our answer

Generating array

0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T

Updating the grammar

- After we calculate generating symbols, we remove all rules that have a symbol that is not generating
- We do not have to explicitly delete any rules
  - We can use a boolean array to indicate which rules are eliminated and which are not eliminated

Calculating Useless Symbols

- We start by calculating generating symbols
  - A symbol is generating if it can derive a string in T^* (zero or more sequence of terminals)
- Then we remove all rules that have a symbol that is not generating
- Then we determine reachable symbols
  - A symbol A is reachable if S can derive a sentential form containing the symbol:

$$S \overset{*}{\Rightarrow} x A y$$

Calculating reachable symbols

1. S is reachable

2. If A → A_1 A_2 … A_k is a grammar rule and A is reachable

⇒ A_1 and A_2 and … and A_k are reachable

Calculating reachable symbols

- Calculation can be done in a way that is similar to how we did generating symbols
- We only consider rules that have not been eliminated in our calculation
- At the end, we have a boolean array indicating which symbols are reachable
- A symbol is not useless if its entries in both the generating array and the reachable array are true, otherwise the symbol is useless

FIRST sets Initialization

0	T	F	F	F	F	F
1	F	F	F	F	F	F
2	F	F	T	F	F	F
3	F	F	F	T	F	F
4	F	F	F	F	T	F
5	F	F	F	F	F	T
6	F	F	F	F	F	F
7	F	F	F	F	F	F
8	F	F	F	F	F	F
9	F	F	F	F	F	F
10	F	F	F	F	F	F
11	F	F	F	F	F	F

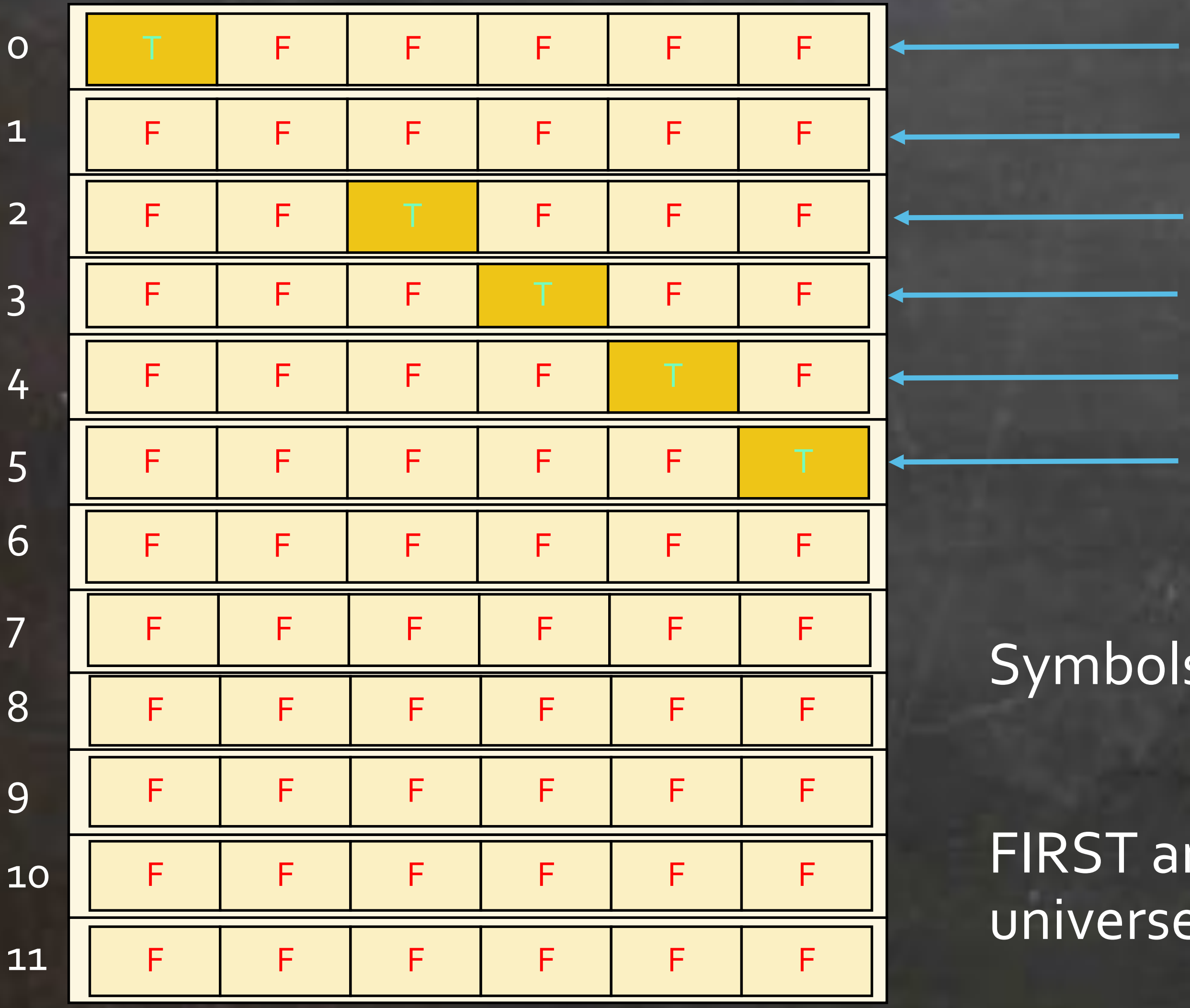
Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

FIRST and FOLLOW universe

0	1	2	3	4	5
“#”	“$”	“b”	“c”	“d”	“e”

FIRST sets Initialization



0	T	F	F	F	F	F
1	F	F	F	F	F	F
2	F	F	T	F	F	F
3	F	F	F	T	F	F
4	F	F	F	F	T	F
5	F	F	F	F	F	T
6	F	F	F	F	F	F
7	F	F	F	F	F	F
8	F	F	F	F	F	F
9	F	F	F	F	F	F
10	F	F	F	F	F	F
11	F	F	F	F	F	F

FIRST(“#”) = { “#” }

FIRST(“$”) = { }

FIRST(“b”) = { “b” }

FIRST(“c”) = { “c” }

FIRST(“d”) = { “d” }

FIRST(“e”) = { “e” }

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
“#”	“$”	“b”	“c”	“d”	“e”	“S”	“A”	“B”	“C”	“D”	“E”

FIRST and FOLLOW universe

0	1	2	3	4	5
“#”	“$”	“b”	“c”	“d”	“e”

FIRST sets Initialization

0	T	F	F	F	F	F
1	F	F	F	F	F	F
2	F	F	T	F	F	F
3	F	F	F	T	F	F
4	F	F	F	F	T	F
5	F	F	F	F	F	T
6	F	F	F	F	F	F
7	F	F	F	F	F	F
8	F	F	F	F	F	F
9	F	F	F	F	F	F
10	F	F	F	F	F	F
11	F	F	F	F	F	F

Symbols

0	1	2	3	4	5	6	7	8	9	10	11
"#"	"$"	"b"	"c"	"d"	"e"	"S"	"A"	"B"	"C"	"D"	"E"

FIRST and FOLLOW universe

0	1	2	3	4	5
"#"	"$"	"b"	"c"	"d"	"e"

FIRST("S") = { }

FIRST("A") = { }

FIRST("B") = { }

FIRST("C") = { }

FIRST("D") = { }

FIRST("E") = { }