Unit 4:
First and Follow Sets - Part One

SCC 312 Compilation
John Vidler
j.vidler@lancaster.ac.uk

Original slide deck by John Mariani

- Given a rule, what are the set of **terminals** that could appear as the first terminal of strings derived by that rule? This is the **FIRST** set.
- Given a rule, what are the set of **terminals** that could appear immediately after a string derived by that rule? This is the **FOLLOW** set.

- In terms of a programming language compiler, we can have a rule for an IF statement.
- if_statement -> IF boolean_expression THEN statement;
- We would expect the FIRST set to contain the IF token, at least.
- If an IF statement can be followed by any programming statement, then the FOLLOW set would consist of all the tokens that can start any programming statement i.e. IF, WHILE etc.

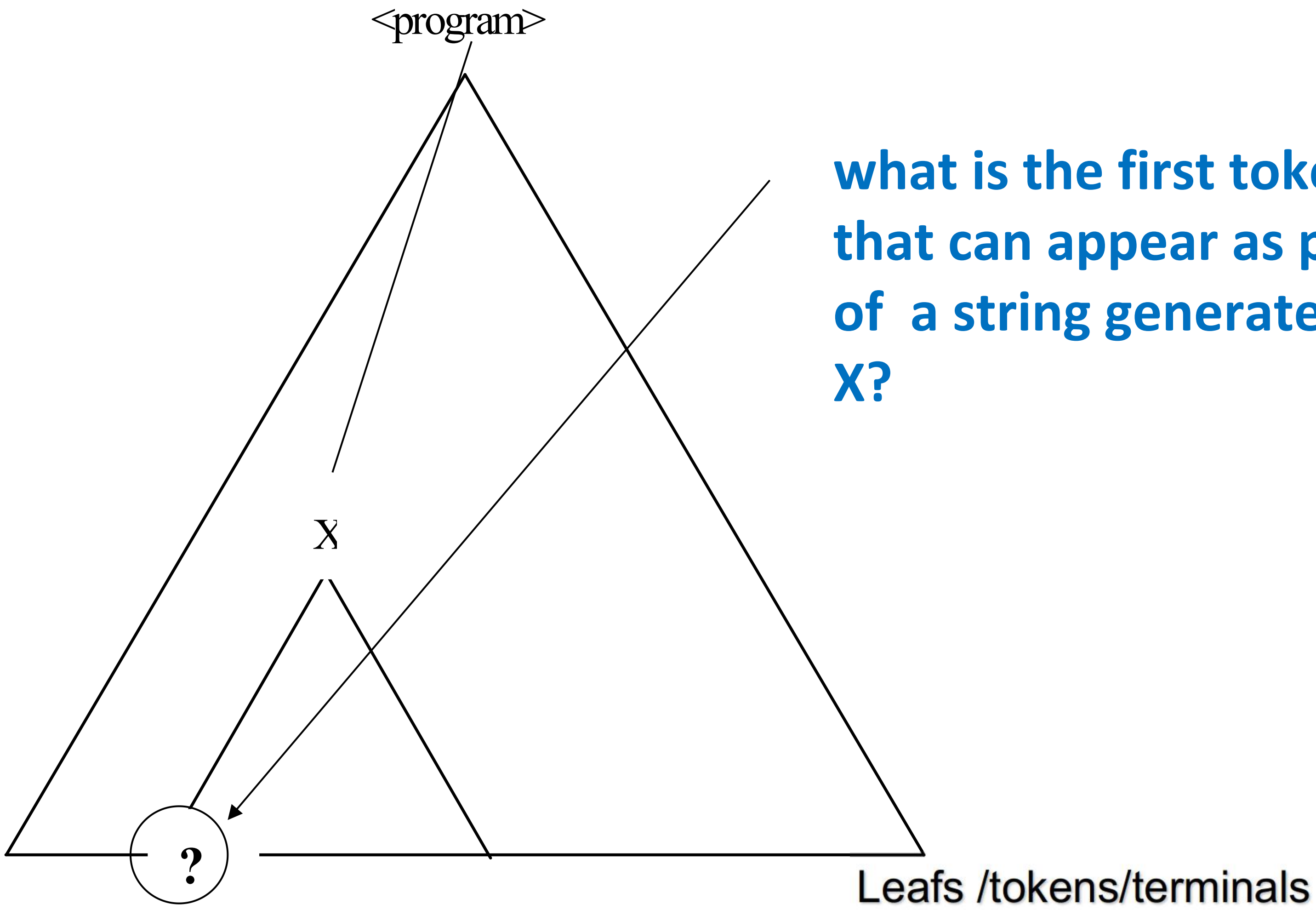
The role of FIRST and FOLLOW sets

- FIRST and FOLLOW sets have a role to play in both the parsing strategies covered in this course, as you shall see.

FIRST sets

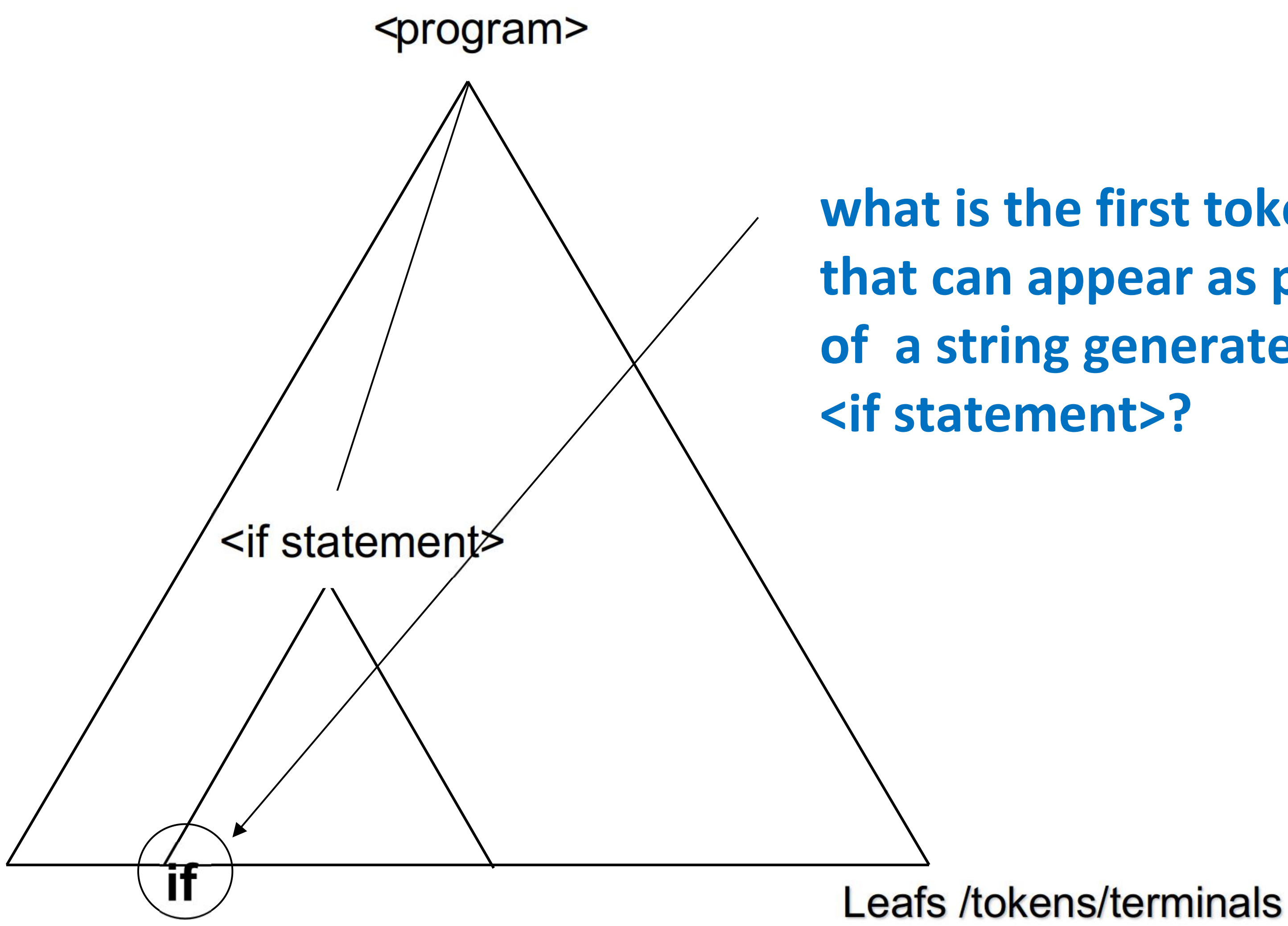
- We start by defining the set FIRST(a) for any string a of terminals and non-terminals
- This is the set of all terminals which could be the first (that is, left-most) terminal of a string derived from a
- Also known as the *Left Terminal Set*

FIRST sets



what is the first token that can appear as part of a string generated by X?

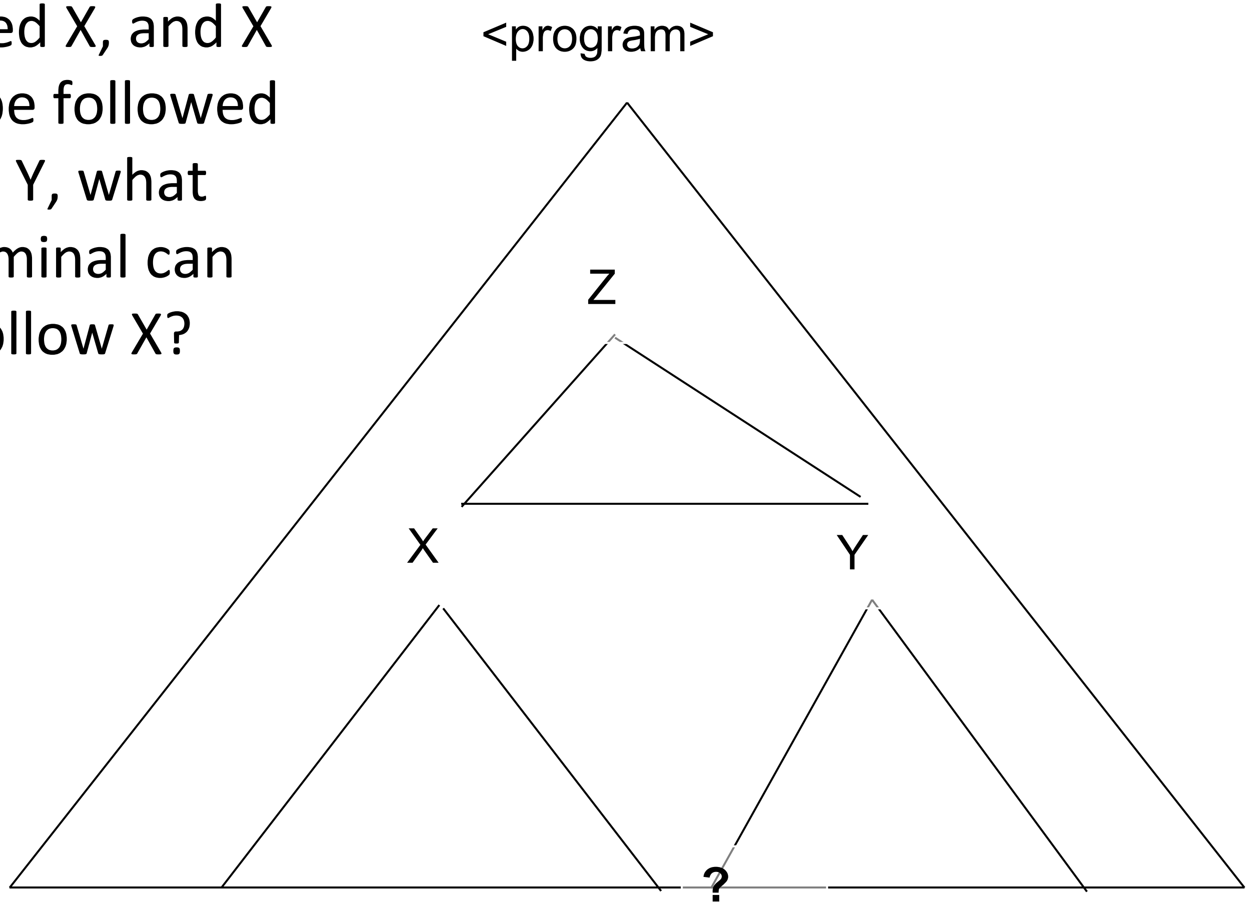
Example



what is the first token that can appear as part of a string generated by <if statement>?

FOLLOW sets

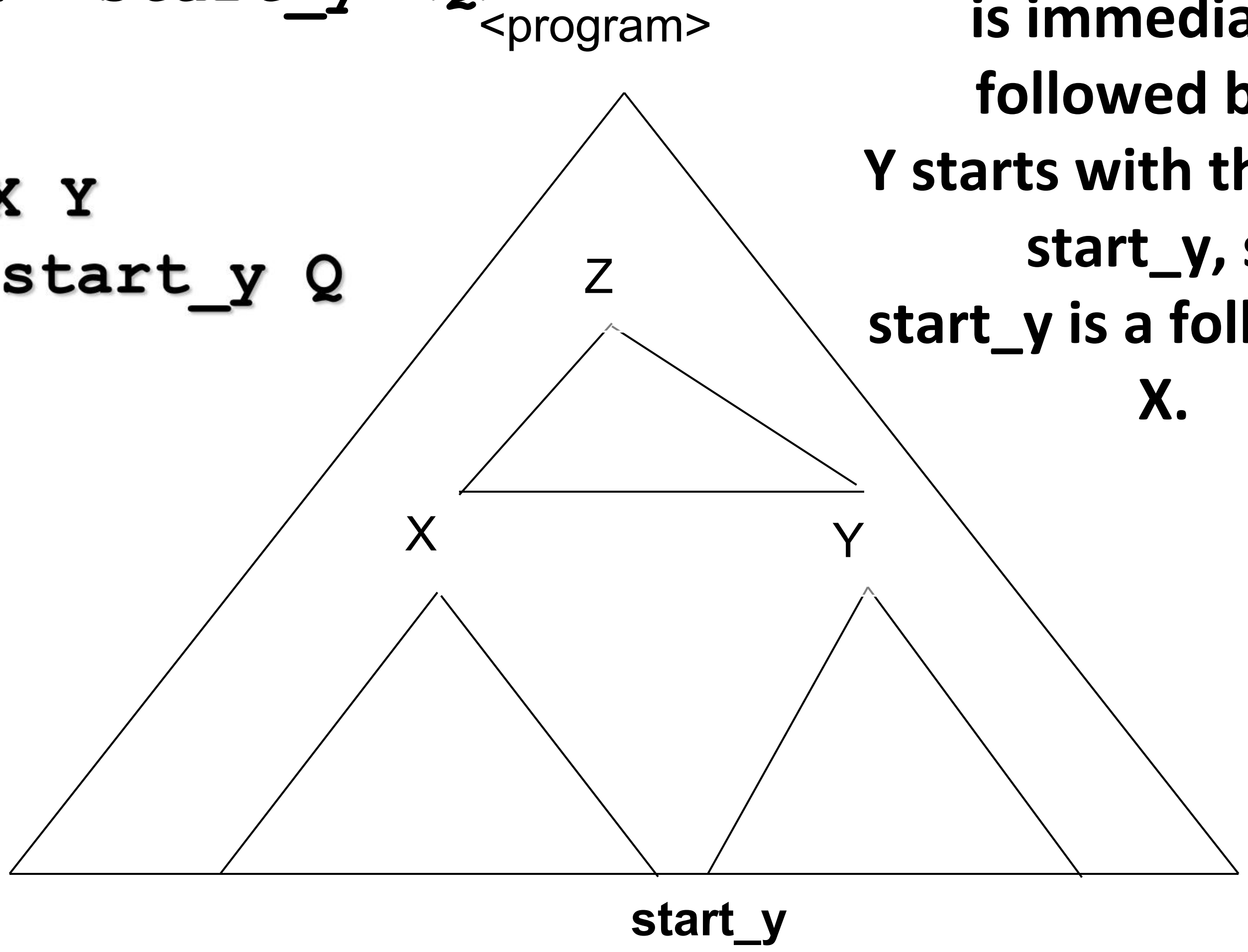
If we have just parsed X, and X can be followed by Y, what terminal can follow X?



Example

```
<Z> ::= <X><Y>
<Y> ::= start_y <Q>
```

```
Z → X Y
Y → start_y Q
```



The grammar tells us X is immediately followed by Y.
Y starts with the token start_y, so start_y is a follower of X.

Null Productions

- **`meal → first_course second_course dessert;`**
- This production rule tells us that a meal consists of a first course, second course, and a dessert.
- **`first_course → SOUP| SALAD|`** ε ;
- ε is a special symbol denoting nothing.
- In other words, a diner may choose to skip the first course.
- This means our grammar contains a null production.
- **`second_course → CHICKEN| FISH| BEEF| LAMB;`**
- So a meal could consist of soup and chicken, or just chicken.

FIRST

- `meal → first_course second_course dessert;`
- `first_course → SOUP| SALAD| ε ;`
- `second_course → CHICKEN| FISH| BEEF| LAMB;`
- FIRST(second_course) = {CHICKEN, FISH, BEEF, LAMB};
- FIRST(first_course) = {SOUP, SALAD, ε?)
- Because the first_course production rule can generate empty, this means the first terminal we may see when processing meal actually belongs to the rule that follows first_course, i.e. second_course.

FIRST

- So we have to add (union) the set for second_course to the set for first_course.
- FIRST(meal) =
  FIRST(first_course) $\cup$ FIRST(second_course) =
  {SOUP, SALAD} $\cup$ {CHICKEN, FISH, BEEF, LAMB} =
  {SOUP, SALAD, CHICKEN, FISH, BEEF, LAMB};

FOLLOW

- `meal → first_course second_course dessert;`
- `first_course → SOUP| SALAD| ε ;`
- `second_course → CHICKEN| FISH| BEEF| LAMB;`
- What follows first_course? The FIRST(second_course), of course!
- FOLLOW(first_course) = {chicken, fish, beef, lamb};

THE END