CS:314 Fall 2024

Section **04**

Recitation **4**

chris.tu@rutgers.edu

Office hours: 2-3pm Thursday CoRE 335

Today's Topics

- LL(1) Grammars
  - First Sets
  - Follow Sets
  - Predict Sets
  - LL(1) Parse Tables

First Sets

For Terminal symbols:
For terminal symbol a, FIRST(a) = {a}. The FIRST set of a terminal is just the terminal itself, since it is the first thing that appears.

For Nonterminal symbols:

- If <N> $\rightarrow$ ε, then add ε to FIRST(<N>)
- If <N> $\rightarrow$ α1 α2 … αn, (α_i are all the symbols on the right side of one single production):
    - **Add FIRST**(α1α2 … αn) to **FIRST**(<N>), where **FIRST**(α1 α2 … αn) is defined as:
    - FIRST(α1) if ε ∉ FIRST(α1)
      because a1 is seen first from <N>'s POV.
    - **Otherwise** (FIRST(α1) – ε) ∪ FIRST(α2 … αn)
      because if a1 is epsilon, then a2 may be seen first from <N>'s POV.

Follow Sets

For Terminal symbols:

A terminal symbol **has no FOLLOW set.**

By definition, **ε does not appear in any FOLLOW set.**

(α, β are symbols that could be terminal or nonterminal)

For Nonterminal symbols:

- For each rule p in the grammar
    - If p is of the form <A> ::= α<B>β, then
        - for each such <B>
            - if ε ∈ FIRST(β)
                - Place {FIRST(β) - ε, FOLLOW(<A>)} in FOLLOW(<B>)
            - else
                - Place {FIRST(β)} in FOLLOW(<B>)
    - If p is of the form <A> ::= α<B>, then
        - Place FOLLOW(<A>) in FOLLOW(<B>)

Example 1

- Consider the following grammar:

```
<decl>       ::= <ID> <decl_tail>

<decl_tail> ::= , <decl>
              | : <ID> ;

<ID>         ::= a | b | c
```

- **Show the LL(1) parse table, making First and Follow sets as needed.**

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

First(<decl>) = {a,b,c}

First(<decl_tail>) = {“,” , “:” }

First(<ID>) = {a,b,c}

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

First Sets

- Start by setting each First set to ∅, then iterate through the production rules.

First(<decl>) = {}
First(<decl_tail>) = {}
First(<ID>) = {}

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

First Sets

- We can add First(<ID>) to First(<decl>), since <decl> ::= <ID> <decl_tail>
  - Since First(<ID>) does not contain epsilon, we don't care about <decl_tail> in this rule.

First(`<decl>`) = {}
First(`<decl_tail>`) = {}
First(`<ID>`) = {}

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

- We can add "," to First(<decl_tail>), since <decl_tail> ::= , <decl>

First(<decl>) = {}
First(<decl_tail>) = {","}
First(<ID>) = {}

First Sets

```
<decl>       ::= <ID> <decl_tail>
<decl_tail>  ::= , <decl>
               | : <ID> ;
<ID>         ::= a | b | c
```

- We can add ":" to First(<decl_tail>), since <decl_tail> ::= : <ID> ;

First(`<decl>`) = {}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {}

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

- We can add "a" to First(<ID>), since <ID> ::= a

First(`<decl>`) = {}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {"a"}

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

First Sets

- We can add "a" to First(<ID>), since <ID> ::= b

First(`<decl>`) = {}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {"a", "b"}

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

- We can add "a" to First(<ID>), since <ID> ::= c

First(`<decl>`) = {}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {"a", "b", "c"}

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

First Sets

- We're not done yet!
- We need to iterate through all the rules again until our sets don't change.

First(`<decl>`) = {}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {"a", "b", "c"}

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

- We can add First(<ID>) to First(<decl>), since <decl> ::= <ID> <decl_tail>
    - Since First(<ID>) does not contain epsilon, we don't care about <decl_tail> in this rule.

First(<decl>) = {"a", "b", "c"}
First(<decl_tail>) = {",", ":"}
First(<ID>) = {"a", "b", "c"}

First Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
```

- At this point, if we perform a full iteration across **all** production rules, we will see no further changes to our First sets.

First(`<decl>`) = {"a", "b", "c"}
First(`<decl_tail>`) = {",", ":"}
First(`<ID>`) = {"a", "b", "c"}

Predict Sets

Define *PREDICT*(<A> ::= δ) for rule <A> ::= δ

$$PREDICT(\langle A\rangle ::= \delta) = \begin{cases} FIRST(\delta) - \{\varepsilon\} \cup \text{Follow}(\langle A\rangle), & \text{if } \varepsilon \in FIRST(\delta) \\ FIRST(\delta), & \text{otherwise} \end{cases}$$

Predict Sets

- None of the First sets contain epsilon.
- So we don't need Follow sets.

```
<decl>        ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
               | : <ID> ;
<ID>          ::= a | b | c
First(<decl>) = {"a", "b", "c"}
First(<decl_tail>) = {",", ":"}
First(<ID>) = {"a", "b", "c"}
```

Predict(<decl> ::= `<ID> <decl_tail>`) = {“a”, “b”, “c”}

Predict(`<decl_tail> ::= , <decl>`) = {“,”}

Predict(`<decl_tail> ::= : <ID> ;`) = {“:”}

Predict(`<ID> ::= a`) = ?

Predict(`<ID> ::= b`) = ?

Predict(`<ID> ::= c`) = ?

Predict Sets

```
<decl>      ::= <ID> <decl_tail>
<decl_tail> ::= , <decl>
              | : <ID> ;
<ID>        ::= a | b | c
First(<decl>) = {"a", "b", "c"}
First(<decl_tail>) = {",", ":"}
First(<ID>) = {"a", "b", "c"}
```

- None of the First sets contain epsilon.
- So we don't need Follow sets.

Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>			a	b	c

- Our parse table must have one row for every non-terminal.
- It has one column for each terminal symbol that appears in the Predict sets.

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>					
<ID>					

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>				
<ID>					

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>					

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {"a", "b", "c"}
Predict(<decl_tail> ::= , <decl>) = {","}
Predict(<decl_tail> ::= : <ID> ;) = {":"}
Predict(<ID> ::= a) = {"a"}
Predict(<ID> ::= b) = {"b"}
Predict(<ID> ::= c) = {"c"}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>			a	b	c

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>			a	b	c

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>			a	b	c

Parse Table

```
Predict(<decl> ::= <ID> <decl_tail>) = {“a”, “b”, “c”}
Predict(<decl_tail> ::= , <decl>) = {“,”}
Predict(<decl_tail> ::= : <ID> ;) = {“:”}
Predict(<ID> ::= a) = {“a”}
Predict(<ID> ::= b) = {“b”}
Predict(<ID> ::= c) = {“c”}
```

	,	:	a	b	c
<decl>			<ID> <decl_tail>	<ID> <decl_tail>	<ID> <decl_tail>
<decl_tail>	, <decl>	: <ID> ;			
<ID>			a	b	c

- The final LL(1) parse table for the grammar is shown above.
- Since there are no conflicts in this table, the grammar is indeed LL(1).
- No Conflicts means that

For each non-terminal and terminal combination in the parse table, there is exactly one valid production rule.

Another Grammar

Suppose we rewrite the previous grammar like so:

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c
```

This generates the **same language**, but is the grammar still **LL(1)**?
Show the parse table, after making First, Follow, and Predict sets.

Another Grammar – First Sets

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c
```

- First(<decl>) = {“a”,”b”,”c”}
- First(<mid>) = {ε, “,”}
- First(<tail>) = {“:”}
- First(<id>) = {“a”,”b”,”c”}

Another Grammar - First Sets

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c
```

- First(<decl>) = {"a", "b", "c"}
- First(<mid>) = {ε, ","}
- First(<tail>) = {":"}
- First(<id>) = {"a", "b", "c"}

- **Since epsilon appears in our First sets, we'll need Follow sets.**

Another Grammar – Follow Sets

- Follow(<decl>) = {eof}
- Follow(<mid>) = {":", "," }
- Follow(<tail>) = {eof}
- Follow(<id>) = {"," , ";", ":" }

```
<decl>  ::= <id> <mid> <tail>
<mid>   ::= <mid> , <id>
          | ε
<tail>  ::= : <id> ;
<id>    ::= a | b | c
```

```
First(<decl>) = {"a", "b", "c"}
First(<mid>) = {",", ε}
First(<tail>) = {":"}
First(<id>) = {"a", "b", "c"}
```

Another Grammar – Follow Sets

- Follow(<decl>) = {eof}
- Follow(<mid>) = {":", ","}
- Follow(<tail>) = {eof}
- Follow(<id>) = {",", ":", ";"}

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c

First(<decl>) = {"a", "b", "c"}
First(<mid>) = {",", ε}
First(<tail>) = {":"}
First(<id>) = {"a", "b", "c"}
```

Another Grammar – Predict Sets

- Predict(<decl> ::= <id> <mid> <tail>) = `{"a", "b", "c"}`
- Predict(<mid> ::= <mid> , <id>) = {`","`}
- Predict(<mid> ::= ε) = `{":", ","}` (Follow(<mid>))
- Predict(<tail> ::= : <id> ;) = `{":"}`
- Predict(<id> ::= a) = “a”
- Predict(<id> ::= b) = “b”
- Predict(<id> ::= c) = “c”

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c

First(<decl>) = {"a", "b", "c"}
First(<mid>) = {",", ε}
First(<tail>) = {":"}
First(<id>) = {"a", "b", "c"}
Follow(<decl>) = {eof}
Follow(<mid>) = {":", ","}
Follow(<tail>) = {eof}
Follow(<id>) = {",", ":", ";"}
```

Another Grammar - Predict Sets

- Predict(<decl> ::= <id> <mid> <tail>) = {"a", "b", "c"}
- Predict(<mid> ::= <mid> , <id>) = {","}
- Predict(<mid> ::= ε) = {":", ","}
  - For this one we just used Follow(<mid>)
- Predict(<tail> ::= : <id> ;) = {":"}
- Predict(<id> ::= a) = {"a"}
- Predict(<id> ::= b) = {"b"}
- Predict(<id> ::= c) = {"c"}

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c

First(<decl>) = {"a", "b", "c"}
First(<mid>) = {",", ε}
First(<tail>) = {":"}
First(<id>) = {"a", "b", "c"}
Follow(<decl>) = {eof}
Follow(<mid>) = {":", ","}
Follow(<tail>) = {eof}
Follow(<id>) = {",", ":", ";"}
```

Another Grammar - Parse Table

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c
```

Predict(<decl> ::= <id> <mid> <tail>) = {"a", "b", "c"}
Predict(<mid> ::= <mid> , <id>) = {","}
Predict(<mid> ::= ε) = {":", ","}
Predict(<tail> ::= : <id> ;) = {":"}
Predict(<id> ::= a) = {"a"}
Predict(<id> ::= b) = {"b"}
Predict(<id> ::= c) = {"c"}

	,	:	a	b	c
<decl>			<id> <mid> <tail>	<id> <mid> <tail>	<id> <mid> <tail>
<mid>	<mid> , <id> ε	ε			
<tail>		: <id> ;			
<id>			a	b	c

Another Grammar – Parse Table

```
<decl> ::= <id> <mid> <tail>
<mid>  ::= <mid> , <id>
         | ε
<tail> ::= : <id> ;
<id>   ::= a | b | c
```

Predict(<decl> ::= <id> <mid> <tail>) = {"a", "b", "c"}
Predict(<mid> ::= <mid> , <id>) = {","}
Predict(<mid> ::= ε) = {":", ","}
Predict(<tail> ::= : <id> ;) = {":"}
Predict(<id> ::= a) = {"a"}
Predict(<id> ::= b) = {"b"}
Predict(<id> ::= c) = {"c"}

	,	:	a	b	c
<decl>			<id> <mid> <tail>		
<mid>	<mid> , <id> ε	ε			
<tail>		: <id> ;			
<id>			a	b	c

Another Grammar - Parse Table

- This grammar is **not** LL(1)!
- The <mid> non-terminal encounters a conflict when we see a comma.
- Even with 1 symbol of lookahead, we cannot decide between its two rules.
  - In fact, this isn't LL at all; no finite number of lookahead symbols would be sufficient.
  - LL(1): No Left recursion; No backtrack

	,	:	a	b	c
<decl>			<id> <mid> <tail>		
<mid>	<mid> , <id> ε	ε			
<tail>		: <id> ;			
<id>			a	b	c

Proving LL(1)

A Grammar is LL(1) iff for any non-terminal <A> with at least two production rules, when looking at any two distinct rules in the form of "<A> ::= α and <A> ::= β", it follows

$$\text{PREDICT}(\langle A\rangle ::= \alpha) \cap \text{PREDICT}(\langle A\rangle ::= \beta) = \varnothing$$

1. Make the predict set for every nonterminal production rule (following the guide on slide 17)
2. For any nonterminal with two or more production rules (or OR cases), <u>make separate predict sets.</u>
3. If the **predict sets** of the **distinct and different rules** for the **same nonterminal** share a symbol, the grammar is NOT LL(1)

IFF for any nonterminal with two or more distinct rules:
If the **predict sets** of the **distinct and different rules** for the **nonterminal** do not overlap/do not share symbols

e.g. $\text{PREDICT}(\langle A \rangle ::= \alpha) \cap \text{PREDICT}(\langle A \rangle ::= \beta) = \varnothing$

the grammar is indeed LL(1).