Top-Down Parsing: Implementation & Limitations

Predicting with Lookahead

- The example we did last time suggests how to implement Predict.
- We were able to determine the correct rule to use just by looking at two pieces of information: the **current nonterminal** to expand, and the **next symbol** of input.
- This might not work for more complicated grammars, but we'll develop this idea and see how well it works.
- In our pseudocode, we allowed the Predict function to depend on the whole "derivedString" but we will work with a simpler idea.
- We will develop **Predict(current nonterminal, next symbol)**.

Implementing Predict

- Given a nonterminal "A" (on top of the stack) and a terminal "a" (at the front of unread input), we need to predict the next rule to apply.
- The terminal "a" is the next unmatched symbol of input, so we want to expand "A" into something that *starts with* "a".
- Suppose A can *derive* a string that starts with "a". Then any rule that appears as the *first step* of such a derivation could be valid to apply.
- We introduce some notation: **First(A)** is the set of all *terminal* symbols such that A can derive a string that starts with the terminal symbol.
- If "a" is in First(A), then Predict should try to find an appropriate rule.

Implementing Predict

- Given a nonterminal "A" (on top of the stack) and a terminal "a" (at the front of unread input), we need to predict the next rule to apply.
- If "a" is in First(A), there should be at least one valid rule to try.
- What if "a" is not in First(A)? Do we give up the parse?
- **No.** Consider the possibility that $A \Rightarrow^* \varepsilon$.
  - That is, A can be "deleted" (replaced by the empty string).
- If the next thing on the stack *after* A is "a", and we can "delete" A, then the parse might still be possible to complete.
- We will say **A** is *nullable* if $A \Rightarrow^* \varepsilon$ and say that **Nullable(A)** is true.

Implementing Predict

- Given a nonterminal "A" (on top of the stack) and a terminal "a" (at the front of unread input), we need to predict the next rule to apply.
- If "a" is in First(A), there should be at least one valid rule to try.
- If "a" is not in First(A), consider whether Nullable(A) is true.
- If A is nullable, it might be the case that "a" can *follow* (appear after) A in a derivation. In this case, we should apply rules that "delete" A.
- We define **Follow(A)** to be the set of *terminal* symbols that can possibly follow A in a derivation starting from the start symbol.
- If Nullable(A) is true and "a" is in Follow(A), then Predict should try to find a rule that either "deletes" A, or works towards this goal.

Implementing Predict

- If "A" is the nonterminal on top of the stack, there are exactly three possibilities:

1. The next input terminal is in First(A).
2. Nullable(A) is true, and the next input terminal is in Follow(A).
3. The parse is impossible to complete.
    - Why? If 1 is false, there is no sequence of rules that can expand A into something that starts with the next input terminal. We need to get rid of A.
    - If 2 is also false, either Nullable(A) is false (so we can't get rid of A) or the next input terminal cannot possibly follow A in a derivation (so getting rid of A would leave us with a mismatch between terminals).

Implementing Predict

- If "A" is the nonterminal on top of the stack, and "a" is the next input terminal, there are two *valid* possibilities (and one error case).

1. The next input terminal "a" is in First(A).
2. Nullable(A) is true, and the next input terminal "a" is in Follow(A).

- In these two cases, how should Predict find a rule to use?
  - Let's not worry about the problem of choosing between *multiple valid rules.* We'll just try to find at least one rule that works.
- In Case 1, look for rules that expand A, and have "a" at the start of the right hand side…?

Implementing Predict

- If "A" is the nonterminal on top of the stack, and "a" is the next input terminal, there are two *valid* possibilities (and one error case).

1. The next input terminal "a" is in First(A).
2. Nullable(A) is true, and the next input terminal "a" is in Follow(A).

- In Case 1, look for rules that expand A, and have "a" at the start of the right hand side…?
- This doesn't cover all possibilities. Consider a scenario like this:

$$A \rightarrow CBC \qquad B \rightarrow CCa \qquad C \rightarrow \varepsilon$$

- This set of rules still implies "a" is in First(A)!

First of a String

- It is not enough to just define First for nonterminals.
- We want to be able to look at the *right hand side of a rule* and determine whether a particular terminal symbol can appear "first" in *anything derived from that right hand side.*

$$A \rightarrow CBC \qquad B \rightarrow CCa \qquad C \rightarrow \varepsilon$$

- For example, we want to be able to say that "a" is in First(CBC) because $CBC \Rightarrow BC \Rightarrow CCaC \Rightarrow CaC \Rightarrow aC$.
- Define **First(α)**, where α can be any sequence of *terminals and nonterminals*, to be the set of *terminal* symbols that can appear first in anything derived from α.

Implementing Predict

- If "A" is the nonterminal on top of the stack, and "a" is the next input terminal, there are two *valid* possibilities (and one error case).

1. The next input terminal "a" is in First(A).
2. Nullable(A) is true, and the next input terminal "a" is in Follow(A).

- In Case 1, look for rules of the form A → α where "a" is in First(α).
- In Case 2, look for rules of the form A → ε ...?
- We have a similar problem: it might be complicated to "nullify" A.

A → BCD B → ε C → DE D → ε E → ε

- Similarly to First, we can define **Nullable(α)** for a string α.

Implementing Predict

- If "A" is the nonterminal on top of the stack, and "a" is the next input terminal, there are two *valid* possibilities (and one error case).

1. The next input terminal "a" is in First(A).
2. Nullable(A) is true, and the next input terminal "a" is in Follow(A).

- In Case 1, look for rules of the form A → α where "a" is in First(α).
- In Case 2, look for rules of the form A → α where Nullable(α) is true.
- If we are in neither case, or no rule is found, the parse is impossible to complete and we return "null" (no rule).

Implementing Nullable, First, and Follow

- If we have algorithms for computing Nullable, First, and Follow, then Predict is straightforward: loop over the rules in the grammar and check the conditions on the previous slide.
- However, computing these is a little tricky.

$$A \rightarrow B \qquad B \rightarrow A \qquad B \rightarrow \varepsilon$$

- Consider Nullable(A). If you tried to compute this recursively, you might get stuck in an infinite loop of "A is nullable if B is nullable if A is nullable if B is nullable…" depending on the order in which you process the rules.

Implementing Nullable

- We will use a *fixed point algorithm* to avoid this infinite recursion.
- We compute Nullable(B) for *every nonterminal B* at the same time.
  - Iterate through all the rules and figure out which nonterminals are "directly" nullable, i.e., there is a rule B → ε.
  - On the next iteration, figure out which nonterminals can derive a string of nonterminals that are all known to be nullable. That is, there is a rule B → β and every symbol in the right hand side β was previously found to be nullable.
  - Repeat until we reach a "fixed point": We do an iteration but we gain no new information about which nonterminals are nullable.
- Nullable(β) is true if and only if every symbol in β is nullable. This can be computed easily using Nullable for nonterminals.

Computing Nullable: Example

A → BD   A → CC   B → b   C → DE   D → ε   E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?

Computing Nullable: Example

A → BD   A → CC   B → b   C → DE   D → ε   E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?
1	?	?	?	True	True

Computing Nullable: Example

A → BD　　A → CC　　B → b　　C → DE　　D → ε　　E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?
1	?	?	?	True	True
2	?	?	True	True	True

Computing Nullable: Example

A → BD   A → CC   B → b   C → DE   D → ε   E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?
1	?	?	?	True	True
2	?	?	True	True	True
3	True	?	True	True	True

Computing Nullable: Example

A → BD   A → CC   B → b   C → DE   D → ε   E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?
1	?	?	?	True	True
2	?	?	True	True	True
3	True	?	True	True	True
4	True	?	True	True	True

Computing Nullable: Example

A → BD   A → CC   B → b   C → DE   D → ε   E → ε

Iteration	Nullable(A)	Nullable(B)	Nullable(C)	Nullable(D)	Nullable(E)
Start	?	?	?	?	?
1	?	?	?	True	True
2	?	?	True	True	True
3	True	?	True	True	True
4	True	?	True	True	True
End	True	False	True	True	True

Implementing First

- We start with First for nonterminals.
- Like Nullable, we compute First(B) for every nonterminal B at the same time, using a fixed point algorithm.
  - For each nonterminal B, and each rule B → β, loop over the symbols in β.
    1. If the current symbol in the loop is a terminal "b", add "b" to First(B) and stop the loop.
    2. If it is a nonterminal C, add everything in First(C) to First(B). Then, if Nullable(C) is *false*, stop the loop. Otherwise, continue the loop and examine the next symbol in β.
- Idea: β could start with a bunch of nullable nonterminals. We process all of these until we find either a terminal, or a nonterminal that is not nullable. At that point, any further symbols in β are irrelevant.

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø U First(B)	Ø	Ø	Ø

B is nullable, so take the union with First(B) and continue

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø U First(C)	Ø	Ø	Ø

B is nullable, so take the union with First(B) and continue

C is nullable, so take the union with First(C) and continue

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø U First(D)	Ø	Ø	Ø

B is nullable, so take the union with First(B) and continue

C is nullable, so take the union with First(C) and continue

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	∅	∅	∅

B is nullable, so take the union with First(B) and continue

C is nullable, so take the union with First(C) and continue

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	**{b}**	∅	∅

Right hand side starts with a terminal "b", add "b" to First(B) and stop

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	∅	∅

Nothing happens

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	Ø ∪ First(C)	Ø

C is nullable, so take the union with First(C) and continue

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	∅ ∪ {c}	∅

C is nullable, so take the union with First(C) (from previous step) and continue

Next is the terminal "c", add it to the set and stop

Computing First: Example

A → BCD　　B → b　　B → ε　　C → Ccb　　C → De　　C → ε　　D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	**{c}**	Ø

C is nullable, so take the union with First(C) (from previous step) and continue

Next is the terminal "c", add it to the set and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	**{c}** U First(D)	∅

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	**{c}**	∅

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	∅

Nothing happens

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	{c}	**{d}**

Right hand side starts with a terminal "d", add "d" to First(D) and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}

Right hand side starts with a terminal "d", add "d" to First(D) and stop

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	{c}	{d}
2	**{b,c,d}**	{b}	{c}	{d}

B is nullable, so take the union with First(B) and continue

C is nullable, so take the union with First(C) and continue

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c}	{d}

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c}	{d}

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	{c}	{d}
2	{b,c,d}	{b}	{c}	{d}

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	Ø	Ø	Ø	Ø
1	Ø	{b}	{c}	{d}
2	{b,c,d}	{b}	**{c,d}**	{d}

D is not nullable, so take the union with First(D) and stop

Computing First: Example

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c,d}	{d}

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c,d}	{d}

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c,d}	{d}

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c,d}	{d}
3	{b,c,d}	{b}	{c,d}	{d}

Nothing new happens on the third iteration through the rules

Computing First: Example

A → BCD   B → b   B → ε   C → Ccb   C → De   C → ε   D → d

Iteration	First(A)	First(B)	First(C)	First(D)
Start	∅	∅	∅	∅
1	∅	{b}	{c}	{d}
2	{b,c,d}	{b}	{c,d}	{d}
3	{b,c,d}	{b}	{c,d}	{d}
End	{b,c,d}	{b}	{c,d}	{d}

Implementing First

- Computing First of a string is exactly the same process as the inner loop for First of a nonterminal.

1. Start with First(β) = ∅.
2. Loop over the symbols in β.
   i. If the current symbol is a terminal, add it to First(β) and stop.
   ii. If it's a nonterminal C, and the nonterminal is not nullable, add First(C) to First(β) and stop.
   iii. If it's a nullable nonterminal C, add First(C) to First(β), then continue the loop to the next symbol in β.

A → BCD B → b B → ε C → Ccb C → De C → ε D → d

- First(BCD) = First(B) ∪ First(C) ∪ First(D), First(CcB) = First(C) ∪ {c}.

Implementing Follow

- This is the trickiest one.
- We use the same strategy: Compute Follow(B) for all nonterminals B at once using a single fixed point algorithm.
- The basic idea is to look at the right hand sides of rules, and find occurrences of nonterminals.
- If we find a nonterminal B on the right hand side of a rule, then we know everything in First(whatever string comes after B) can follow B.
- But there's also a special case: When B is at the far right of a rule and has nothing after it, OR when everything after B is nullable.

Follow Special Case

- If a nonterminal B appears at the far right end of a rule, can we conclude anything about what can follow B?
- **Yes.** In this case, anything that can follow the *left hand side* of the rule can follow B.
- Example: S → SABC, S → ε, A → a, A → ε, B → b, C → c, C → ε
- Follow(C) contains Follow(S), because if the string β can follow S in a derivation, then it can also follow C since $S\beta \Rightarrow SABC\beta$.
- Follow(B) also contains Follow(S), since C is nullable, so $S\beta \Rightarrow SABC\beta \Rightarrow SAB\beta$. So this case also applies when everything that appears after a nonterminal is nullable.

Implementing Follow

- For each nonterminal B and each B → β, loop over each symbol in β.
- If the current symbol is a terminal, ignore it and continue.
- If the current symbol is a nonterminal C, let γ denote the rest of β that comes *after* C.
  - Add everything in First(γ) to Follow(C).
  - Additionally, if Nullable(γ) is true, add everything in Follow(B) to Follow(C) (where B is the left hand side of the current rule). Note that this applies in the case where γ = ε, meaning C was the last symbol in β.
- The above describes *one iteration* of a fixed-point algorithm. Repeat this process until no new information about Follow sets is obtained.

Computing Follow: Example

- S → SABC   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅

Computing Follow: Example

- S → **S**ABC S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	∅	∅	∅	∅

We consider First(ABC).

"a" is in First(ABC), but so is "b" since A is nullable and First(B) = {b}.

So First(ABC) = {a,b}.

Computing Follow: Example

- S → **S**ABC   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	Ø	Ø	Ø	Ø
1	**{a,b}**	Ø	Ø	Ø

Add First(ABC) to Follow(S).

Nullable(ABC) is false, so we move on.

Computing Follow: Example

- S → SABC   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	∅	∅	∅

Now consider First(BC).

Since B is not nullable, we have First(BC) = {b}.

Computing Follow: Example

- S → SABC S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	**{b}**	∅	∅

We add First(BC) to Follow(A).

Nullable(BC) is false, so we move on.

Computing Follow: Example

- S → SA**B**C S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	{b}	∅	∅

Consider First(C). This is just {c}.

Computing Follow: Example

- S → SA**B**C S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	{b}	**{c}**	∅

We add First(C) to Follow(B).

But Nullable(C) is true, so we are not done.

Computing Follow: Example

- S → SA**B**C S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	{b}	**{a,b,c}**	∅

Since we are processing S → SABC, we add Follow(S) to Follow(B) since S is the left hand side of the rule.

Computing Follow: Example

- S → SA**B**C   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	Ø	Ø	Ø	Ø
1	{a,b}	{b}	{a,b,c}	Ø

Now we are done with handling the nonterminal B.

Computing Follow: Example

- S → SABC   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	{b}	{a,b,c}	∅

There is nothing after C in this rule, so we don't consider the First set of anything.

Computing Follow: Example

- S → SAB**C**   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	Ø	Ø	Ø	Ø
1	{a,b}	{b}	{a,b,c}	**{a,b}**

But since C is at the far right end of the rule, we add Follow(S) to Follow(C).

Computing Follow: Example

- S → SABC   S → ε   A → a   A → ε   B → b   C → c   C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	∅	∅	∅	∅
1	{a,b}	{b}	{a,b,c}	{a,b}

We are done processing the first rule.

None of the other rules have nonterminals on the right hand side, so they are irrelevant.

Computing Follow: Example

- S → SABC S → ε A → a A → ε B → b C → c C → ε
- In this grammar, S → SABC is the only rule where anything interesting happens in the algorithm.

Iteration	Follow(S)	Follow(A)	Follow(B)	Follow(C)
Start	Ø	Ø	Ø	Ø
1	{a,b}	{b}	{a,b,c}	{a,b}
2	{a,b}	{b}	{a,b,c}	{a,b}
End	{a,b}	{b}	{a,b,c}	{a,b}

On the second iteration, nothing ends up changing, so we get this result.

Predict Tables

- It is common to implement Predict as a lookup table, where you look up a (nonterminal, terminal) pair and it tells you which rules are valid.
- To fill out this table, loop over all productions A → α in the grammar.
  - For each symbol "a" in First(α), add A → α to Predict(A, a).
  - If Nullable(α) is true, then for each symbol "a" in Follow(A), add A → α to Predict(A, a).
- Note that each cell of the table is a *set* of rules. The set may be empty (no rule is valid), it may contain one element (unique valid rule) or it may contain multiple elements (multiple choices of rule).
- We'll discuss the "multiple choices of rule" case soon.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'							
S							
C							

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'							
S							
C							

Look at First(⊢S⊣) = {⊢}

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S							
C							

Look at First(⊢S⊣) = {⊢}, add rule 1 to Predict(S', ⊢).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S							
C							

Nullable(⊢S⊣) is false so we continue.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}				
C							

First(aSb) = {a}, so add rule 2 to Predict(S, a).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}				
C							

Nullable(aSb) is false, so continue.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}			{3}	
C							

First(dSe) = {d}, so add rule 3 to Predict(S, d).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}			{3}	
C							

Nullable(dSe) is false, so continue.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}		**{4}**	{3}	
C							

First(C) = {c}, so add rule 4 to Predict(S, c).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S			{2}		{4}	{3}	
C							

Nullable(C) is true, so consider Follow(S).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		**{4}**	{2}	**{4}**	{4}	{3}	**{4}**
C							

Follow(S) = {b,e,⊣}, so add rule {4} for each of those terminals.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C					**{5}**		

First(cC) = {c}, so add rule 5 to Predict(C, c).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C					{5}		

Nullable(cC) is false, so continue.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C					{5}		

First(ε) is empty, so do nothing.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C					{5}		

Nullable(ε) is true, so consider Follow(C).

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		**{6}**		**{6}**	{5}		**{6}**

Follow(C) = {b,e,⊣} so add rule 6 for each of those terminals.

Predict Table Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Predict table complete.

Multiple Choices of Rule

- Consider this very simple (though ambiguous) expression grammar:

  (1) E → E + E   (2) E → 3

- Let's compute the predict table. Nothing is nullable, so just consider First sets.
  - First(E+E) = {3}, so add rule 1 to Predict(E, 3).
  - First(3) = {3}, so add rule 2 to Predict(E, 3)

	+	3
E		{1, 2}

- Notice that this table is totally useless. It says "if the next symbol is 3, it could be either of the two rules in the grammar".
  - In other words, the behaviour of Predict is "figure it out yourself".
- But the rules are not interchangeable. For example, if the string is 3 + 3 + 3, then starting with E → 3 will not work.

LL(1) Parsing

- The technique we have developed is called **LL(1) parsing:**
  - **Left-to-right** scan of the input
  - **Leftmost** derivation is produced
  - **1** symbol of "lookahead" used for prediction
- This works well if each cell in the Predict table contains at most one rule. But there are many grammars where this isn't the case.
- If the Predict table for a grammar has cells with multiple rules, we throw our hands up and say "this grammar is **not LL(1)**, we can't parse it with this technique".
  - Technically, we could use backtracking, but this sacrifices one of the main strong points of LL(1) which is that it is efficient (linear time).

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: **⊢dacbe⊣**

Top of Stack: **S'**

Lookahead: **⊢**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: **⊢dacbe⊣**

Top of Stack: ⊢

Lookahead: ⊢

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢**dacbe⊣**

Top of Stack: **S**

Lookahead: **d**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢**dacbe⊣**
Top of Stack: **d**
Lookahead: **d**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢d**acbe⊣**
Top of Stack: **S**
Lookahead: **a**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢d**acbe⊣**

Top of Stack: **a**

Lookahead: **a**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢da**cbe⊣**

Top of Stack: **S**

Lookahead: **c**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	Match
cbe⊣	⊢da	Sbe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢da**cbe⊣**

Top of Stack: **C**

Lookahead: **c**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	Match
cbe⊣	⊢da	Sbe⊣	Apply (4)
cbe⊣	⊢da	Cbe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢da**cbe⊣**

Top of Stack: **c**

Lookahead: **c**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	Match
cbe⊣	⊢da	Sbe⊣	Apply (4)
cbe⊣	⊢da	Cbe⊣	Apply (5)
cbe⊣	⊢da	cCbe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢dac**be⊣**

Top of Stack: **C**

Lookahead: **b**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	Match
cbe⊣	⊢da	Sbe⊣	Apply (4)
cbe⊣	⊢da	Cbe⊣	Apply (5)
cbe⊣	⊢da	cCbe⊣	Match
be⊣	⊢dac	Cbe⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢dac**be⊣**

Top of Stack: **b**

Lookahead: **b**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
⊢dacbe⊣		⊢S⊣	Match
dacbe⊣	⊢	S⊣	Apply (3)
dacbe⊣	⊢	dSe⊣	Match
acbe⊣	⊢d	Se⊣	Apply (2)
acbe⊣	⊢d	aSbe⊣	Match
cbe⊣	⊢da	Sbe⊣	Apply (4)
cbe⊣	⊢da	Cbe⊣	Apply (5)
cbe⊣	⊢da	cCbe⊣	Match
be⊣	⊢dac	Cbe⊣	Apply (6)
be⊣	⊢dac	be⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢dacb**e⊣**
Top of Stack: **e**
Lookahead: **e**

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
	…		
be⊣	⊢dac	be⊣	Match
e⊣	⊢dacb	e⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢dacbe⊣

Top of Stack: ⊣

Lookahead: ⊣

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
…			
be⊣	⊢dac	be⊣	Match
e⊣	⊢dacb	e⊣	Match
⊣	⊢dacbe	⊣	

LL(1) Parsing: Example

1. S' → ⊢S⊣
2. S → aSb
3. S → dSe
4. S → C
5. C → cC
6. C → ε

	⊢	⊣	a	b	c	d	e
S'	{1}						
S		{4}	{2}	{4}	{4}	{3}	{4}
C		{6}		{6}	{5}		{6}

Input: ⊢dacbe⊣
Top of Stack: (empty)
Lookahead: (none)

Unread	Matched	(T) Stack (B)	Action
⊢dacbe⊣		S'	Apply (1)
…			
be⊣	⊢dac	be⊣	Match
e⊣	⊢dacb	e⊣	Match
⊣	⊢dacbe	⊣	Match
	⊢dacbe⊣		**Accept**

Rules applied (in order): 1, 3, 2, 4, 5, 6

Leftmost derivation:

S' ⇒ ⊢S⊣ ⇒ ⊢dSe⊣ ⇒ ⊢daSbe⊣
⇒ ⊢daCbe⊣ ⇒ ⊢dacCbe⊣
⇒ ⊢dacbe⊣