The Parsing Problem:
Given a grammar $G$, and a string $s$:

1. is $s \in L(G)$?
2. what is a/the/every structure assigned to $s$ in $G$?

- Two basic parsing strategies:
  - Top-down
  - Bottom-up
- Today, we focus on the top-down strategy

Parsing Context-Free Grammars Top-Down

The basic idea:

1. We will try to build a tree $t$ which generates $s$ and which satisfies $G$. If we can, then $s$ is generated by $G$, and moreover with structure $t$. If we can’t, then we will conclude that $s$ is not generated by $G$.
2. We know:
   every string $s$ is in $L(G)$ iff there is some tree $t$ whose yield is $s$ and which
   (a) has as its root label the start symbol in $G$
   (b) for every parent $n$ and list of daughters $d$, $n \rightarrow d$ is a rule in $G$
3. So we know that if $s$ is generated by $G$, then it must have a structure rooted in $S$ (the start symbol of $G$). Thus, we start building a simple tree $t$, with one node, labeled with $S$.
4. We know that a tree can only have terminals at its leaves. So we choose the left-most leaf node in $t$ labeled with a non-terminal $A$, and try to expand it by selecting a production in $G$ $A \rightarrow \gamma$, and updating $t$ so that it has $\gamma$ as daughters of that node $A$. Thus, we predict what the structure of $s$ is.
5. Some of our predictions will be leaves labeled with terminal symbols. These we can scan, matching them against the data ($s$). In general, we will only be concerned with scanning initial segments of the fringe of our tree $t$ against $s$: if the yield of $t$ is $\sigma A \gamma$, where $\sigma$ is a string of terminals, and $A$ is a non-terminal, then we match our predictions against the data by verifying that $\sigma$ is a prefix of $s$.

Example 1 Let $G = (\{S\}, \{a, b\}, S, \{S \rightarrow aSb, S \rightarrow \epsilon\})$

Let us try to parse the string $aabb$.

1. We begin by assuming that the tree is a single node:

   $S$
2. Next, we predict that the left-most leaf node labeled by a non-terminal, $S$, is expanded with the rule $S \rightarrow aSb$:

```
    S
   / \  \
  a   S   b
```

3. We scan the left-most terminal sequence in our tree, $a$, attempting to match it with a sub-sequence of the input, $aabb$

4. Next we predict that the left-most leaf node labeled by a non-terminal, $S$, is expanded with the rule $S \rightarrow aSb$:

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

5. We again scan the left-most terminal sequence in our tree, $aa$, and match it with a subsequence of the input, $aabb$

6. We next predict that the left-most leaf node labeled by a non-terminal, $S$, is expanded with the rule $S \rightarrow \epsilon$:

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

```
    S
   / \  \
  a   S   b
```

```
  a   S
 / \  \
 a   S   b
```

7. We again scan the left-most terminal sequence in our tree, $aabb$, matching it with the input, $aabb$.

8. As we have scanned the complete input, and derived a legitimate tree, we are done!

**Formulating top-down parsing incrementally**

- Our parser minimally needs to keep track of:
  1. what parts of the input have yet to be scanned with our predictions
  2. what non-terminals decorate the fringe of our thus-far predicted tree

- We represent this in terms of a parser state $c = \langle s, \gamma \rangle$, which contains:
  1. a list of input words $s$
2. a list of non-terminal symbols \( \gamma \)

- We can thus re-formulate our top-down parsing strategy more economically:

<table>
<thead>
<tr>
<th>operation</th>
<th>effect</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>( \langle s, A\gamma \rangle \Rightarrow \langle s, \delta\gamma \rangle )</td>
<td>( A \rightarrow \delta \in G )</td>
</tr>
<tr>
<td>scan</td>
<td>( \langle as, a\gamma \rangle \Rightarrow \langle s, \gamma \rangle )</td>
<td></td>
</tr>
</tbody>
</table>

- the initial configuration on a string \( s \) is \( \langle s, S \rangle \), where \( S \) is the start symbol of our grammar
- a final configuration is one in which we’ve scanned the entire string, and have no non-terminal leaves left: \( \langle \epsilon, \epsilon \rangle \)

Example 2 Let \( G = \langle \{S\}, \{a, b\}, S, \{S \rightarrow aSb, S \rightarrow \epsilon\} \rangle \)

Let us try to parse the string \( aabb \).

start: \( \langle aabb, S \rangle \)

predict: \( \langle aabb, aSb \rangle \) (using rule \( S \rightarrow aSb \))

scan: \( \langle abb, Sb \rangle \)

predict: \( \langle abb, aSbb \rangle \) (using rule \( S \rightarrow aSb \))

scan: \( \langle bb, Sbb \rangle \)

predict: \( \langle bb, bb \rangle \) (using rule \( S \rightarrow \epsilon \))

scan: \( \langle b, b \rangle \)

scan: \( \langle \epsilon, \epsilon \rangle \)

- This gives us a way to check whether a string has a parse
- How do we recover that parse?
- We extend our parser state to include a record of which rules were used to rewrite the left-most non-terminal at each step:

<table>
<thead>
<tr>
<th>operation</th>
<th>effect</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>( \langle s, A\gamma, \omega \rangle \Rightarrow \langle s, \delta\gamma, \omega(A, \delta) \rangle )</td>
<td>( A \rightarrow \delta \in G )</td>
</tr>
<tr>
<td>scan</td>
<td>( \langle as, a\gamma, \omega \rangle \Rightarrow \langle s, \gamma, \omega \rangle )</td>
<td></td>
</tr>
</tbody>
</table>

Example 3 Let \( G = \langle \{S\}, \{a, b\}, S, \{S \rightarrow aSb, S \rightarrow \epsilon\} \rangle \)

Let us try to parse the string \( aabb \).

start: \( \langle aabb, S, \epsilon \rangle \)

predict: \( \langle aabb, aSb, (S, aSb) \rangle \) (using rule \( S \rightarrow aSb \))

scan: \( \langle abb, Sb, (S, aSb) \rangle \)
predict:  \( \langle abb, aSbb, (S, aSb)(S, aSb) \rangle \) (using rule \( S \rightarrow aSb \))

scan:  \( \langle bb, Sbb, (S, aSb)(S, aSb) \rangle \)

predict:  \( \langle bb, bb, (S, aSb)(S, aSb)(S, \epsilon) \rangle \) (using rule \( S \rightarrow \epsilon \))

scan:  \( \langle b, b, (S, aSb)(S, aSb)(S, \epsilon) \rangle \)

scan:  \( \langle \epsilon, \epsilon, (S, aSb)(S, aSb)(S, \epsilon) \rangle \)

So \( aabb \) is accepted by our grammar, with a parse tree given as a left-most derivation: \( (S, aSb)(S, aSb)(S, \epsilon) \)