5.3 Applications of Context-Free Grammars

Example 5.2.2: Suppose we are given a context-free grammar $G$ and a string $s$ from the language $L(G)$. We want to determine whether $s$ is a member of $L(G)$. To do this, we can use a technique called recursive descent parsing. The algorithm begins by guessing that the first symbol of $s$ is the start symbol of $G$. If the first symbol of $s$ matches the start symbol, we proceed to the next symbol. If the next symbol matches a terminal symbol, we accept the string. If the next symbol matches a non-terminal symbol, we try to generate a sequence of symbols that matches the recursive descent parse for that non-terminal symbol. If we are unable to generate a sequence of symbols that matches the recursive descent parse for that non-terminal symbol, we reject the string.

Example 5.2.3: Suppose we are given a context-free grammar $G$ and a string $s$. We want to determine whether $s$ is a member of $L(G)$. To do this, we can use a technique called top-down parsing. The algorithm begins by generating a sequence of symbols that matches the start symbol of $G$. If the first symbol of the sequence matches the first symbol of $s$, we proceed to the next symbol. If the next symbol of the sequence matches the next symbol of $s$, we accept the string. If the next symbol of the sequence matches a non-terminal symbol, we try to generate a sequence of symbols that matches the top-down parse for that non-terminal symbol. If we are unable to generate a sequence of symbols that matches the top-down parse for that non-terminal symbol, we reject the string.

Example 5.2.4: Suppose we are given a context-free grammar $G$ and a string $s$. We want to determine whether $s$ is a member of $L(G)$. To do this, we can use a technique called bottom-up parsing. The algorithm begins by generating a sequence of symbols that matches the start symbol of $G$. If the first symbol of the sequence matches the first symbol of $s$, we proceed to the next symbol. If the next symbol of the sequence matches the next symbol of $s$, we accept the string. If the next symbol of the sequence matches a non-terminal symbol, we try to generate a sequence of symbols that matches the bottom-up parse for that non-terminal symbol. If we are unable to generate a sequence of symbols that matches the bottom-up parse for that non-terminal symbol, we reject the string.
2. APPLICATIONS OF CONTEXT-FREE GRAMMARS

1.2. CONTEXT-FREE GRAMMARS AND LANGUAGES
3.3. THE YACC PARSER-GENERATOR

The first line in the Yacc statement for each grammar starts with a capital letter and ends with a punctuation mark, and then continues on the next line with a corresponding punctuation mark.

Yacc is a program that generates parsers for a given grammar. It takes a file specified by the user as input and outputs a parser program in the target language (usually C or C++).

Yacc uses a simple syntax for specifying grammars. Each grammar starts with a statement specifying the start symbol of the grammar, followed by a list of productions (rules). Each production consists of a left-hand side (a terminal symbol or a nonterminal symbol) followed by a colon and a right-hand side (a sequence of terminal symbols and/or nonterminal symbols).

Example: The production statement for the language of arithmetic expressions could be:

```
expression : expression + term |
            term               |
            expression - term |
```

This example shows how Yacc can be used to generate parsers for languages with complex structures.

Yacc is particularly useful for generating parsers for programming languages, where the complexity of the grammar can make hand-written parsers difficult to write and maintain.

Yacc is a powerful tool for generating parsers, but it is not without its limitations. It is not as flexible as some other parser generators and may not be the best choice for all applications. However, its simplicity and efficiency make it a good choice for many situations.
Figure 6.2: HTML Document and Its Parsed Version

1. Document
2. Text
3. Element
4. Element Text
5. Element Text
6. Document Text
7. Document Text

---

HTML is the content of a document, which may be a simple list of text or a more complex structure. It is defined by a set of rules and tags that describe the structure of the document.

There are two main components of HTML: the document itself and the presentation of the document. The document contains the content of the document, while the presentation describes how the content should be displayed.

---

In the example shown in Figure 6.2(a), the HTML document includes some text and a single element.

---

The example shown in Figure 6.2(b) illustrates how the HTML parser interprets the document. The parser examines the document and creates a tree structure that represents the document's structure.

---

The HTML parser then processes the document, applying a set of rules to determine how the document should be displayed. This process is called the rendering process.
However, the title

Processors — don’t under stress

is analogous to the production

of the correct item, for instance.

are now under the production of the controllers. Here is the idea of the values

of the three inputs to the

result in the DDL, the DDI, the DDI.

of the correct item for instance.

You can now get the values for

the values for the ØR.A to the

DDT, the DDI, the DDI.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.

- The 5.15 property.
and the productions:

\[ \begin{align*}
B & \rightarrow \text{Disk} \\
A & \rightarrow \text{Processor, Disk} \\
B & \rightarrow \text{Processor, Disk, Processor} \\
A & \rightarrow \text{Processor, Disk, Processor, Disk} \\
\end{align*} \]

The first production, \[ A \rightarrow B \text{C} \text{D}, \] is needed for C's. The last two may or may not be fair. However, their order is important to the body of the

The productions:

\[ \begin{align*}
C & \rightarrow \text{Processor} \\
B & \rightarrow \text{Processor, Disk} \\
A & \rightarrow \text{Processor, Disk, Processor} \\
\end{align*} \]

Example 5.24: Let us consider how to convert the D/D rule

The production is of the form \( A \rightarrow \text{Processor, Disk} \), replace this production by

The production is of the form \( A \rightarrow \text{Processor, Disk} \), introduce a new variable \( B \)

The production is of the form \( A \rightarrow \text{Processor, Disk} \), introduce a new variable \( B \)

That appears, replace these and replace this production by

3. The production is of the form \( A \rightarrow \text{Processor, Disk} \), introduce a new variable \( B \)

That appears, replace these and replace this production by

4. The production is of the form \( A \rightarrow \text{Processor, Disk} \), introduce a new variable \( B \)

To CG form, apply the rules recursively and recursively convert these new productions

Again, these productions may or may not be kept C's productions but

The production is of the form \( A \rightarrow \) | | B) For the union operator.

220 APPLICATIONS OF CONTEXT-FREE GRAMMARS
We have already seen the diagonalization of CFGs, which is a key concept in understanding the limitations of context-free grammars.

### 5.4 Ambiguity in Grammars and Languages

- **Exercise 5.3.2:** Consider the PDA of Figure 5.15 and a context-free grammar.

- **Exercise 5.4.1:** Add the following lines to the HTML grammar of Figure 5.16:
  
  ```html
  <!-- HTML grammar -->
  ```

  Then, draw a PDA for this grammar.

  - **Exercise 5.4.3:** In Section 5.3.1, we considered the grammar

  
  ```
  ```

  and claimed that we could test for membership in the language by repeatedly

  ```
  SS → S + S |
  S → a |
  ```

  and recorded the number of times 'S' occurred. The number of occurrences of 'S' in this string is the length of the input.

  - **Exercise 5.4.4:** Add the following lines to the HTML grammar of Figure 5.16:

  ```html
  <!-- HTML grammar -->
  ```

  Then, draw a PDA for this grammar.

  - **Exercise 5.4.5:** Consider the context-free language

  ```
  L = \{ w \in \{a, b\}^* | w \text{ has an equal number of } a's \text{ and } b's \}
  ```

  and construct a context-free grammar for this language.

  - **Exercise 5.5.1:** In Section 5.3.1, we considered the language

  ```
  L = \{ a^n b^n | n \geq 0 \}
  ```

  and constructed a context-free grammar for this language.

  - **Exercise 5.5.2:** Consider the context-free language

  ```
  L = \{ a^n b^n c^n | n \geq 0 \}
  ```

  and construct a context-free grammar for this language.

  - **Exercise 5.6.1:** In Section 5.3.1, we considered the language

  ```
  L = \{ a^m b^n c^p | m, n, p \geq 0 \}
  ```

  and constructed a context-free grammar for this language.

  - **Exercise 5.6.2:** Consider the context-free language

  ```
  L = \{ a^n b^n c^n d^n | n \geq 0 \}
  ```

  and construct a context-free grammar for this language.

  - **Exercise 5.7.1:** In Section 5.3.1, we considered the language

  ```
  L = \{ a^n b^n c^m | m, n \geq 0 \}
  ```

  and constructed a context-free grammar for this language.

  - **Exercise 5.7.2:** Consider the context-free language

  ```
  L = \{ a^n b^n c^m d^m | m, n \geq 0 \}
  ```

  and construct a context-free grammar for this language.

- **Exercise 5.8.1:** In Section 5.3.1, we considered the language

  ```
  L = \{ a^n b^n c^m d^m | m, n \geq 0 \}
  ```

  and constructed a context-free grammar for this language.

  - **Exercise 5.8.2:** Consider the context-free language

  ```
  L = \{ a^n b^n c^m d^m e^m | m, n \geq 0 \}
  ```

  and construct a context-free grammar for this language.

- **Exercise 5.9.1:** In Section 5.3.1, we considered the language

  ```
  L = \{ a^n b^n c^m d^m e^m f^m | m, n \geq 0 \}
  ```

  and constructed a context-free grammar for this language.