Using recursion to define objects

We can use recursion to define functions:

The factorial function can be defined as:

\[ n! = \begin{cases} 
1 & \text{for } n = 0 \\
n \cdot (n - 1)! & \text{otherwise} 
\end{cases} \]

This gives us a way of computing the function for any value of \( n \).

Recursive sets

Some sets are most naturally specified with recursive definitions.

A recursive definition of a set shows how to construct elements in the set by putting together simpler elements.

Example: balanced parentheses

\((()())\) is balanced

\(())\) and \((()())\) are not

Recursive objects and structural induction

6.9 - 6.10 in zybooks

factorial

This is all we need to put together the function:

```python
def factorial(n):
    # precondition:  n is a positive integer
    if (n == 0):
        return 1
    else:
        return n * factorial(n-1);
```
Some sets are most naturally specified with recursive definitions.

A recursive definition of a set shows how to construct elements in the set by putting together simpler elements.

Example: balanced parentheses

Basis: The sequence () is properly nested.

Recursive rules: If u and v are properly-nested sequences of parentheses then:
1. (u) is properly nested.
2. uv is properly nested.

Is there a unique way to construct ()(())?
**binary strings**

Let \( B = \{0, 1\} \)

\( B^k \): the set of strings of length \( k \) - \( \{0, 1\}^k \)

The empty string: \( \lambda \)

\( B^0 = \{\lambda\} \)

The set of all strings:

\[
B^* = B^0 \cup B^1 \cup B^2 \ldots
\]

**perfect binary trees**

**Basis:** A single vertex with no edges is a perfect binary tree.

**Recursive rule:** If \( T \) is a perfect binary tree, a new perfect binary tree \( T' \) can be constructed by taking two copies of \( T \), adding a new vertex \( v \) and adding edges between \( v \) and the roots of each copy of \( T \). The new vertex \( v \) is the root of \( T' \).

---

**binary strings**

The set of all strings:

\[
B^* = B^0 \cup B^1 \cup B^2 \ldots
\]

This set can be defined recursively:

**Base case:** \( \lambda \in B^* \)

**Recursive rule:** if \( x \in B^* \) then

- \( x0 \in B^* \)
- \( x1 \in B^* \)
We can use induction to prove properties of recursively defined objects.

This is called structural induction.

As an example, we’ll prove the following:

**Theorem:** Properly nested strings of left and right parentheses are balanced.

A string of parentheses $x$ is called **balanced** if $\text{left}[x] = \text{right}[x]$, where $\text{left}[x]/\text{right}[x]$ is the number of left/right parentheses in $x$.

**Proof.**

By induction.

**Base case:** $\varepsilon$ is properly nested. $\text{left}[\varepsilon] = \text{right}[\varepsilon] = 1$.

**Inductive step:** If $x$ is a string of properly nested parentheses then $x$ was constructed by applying a sequence of recursive rules starting with the string $\varepsilon$. We consider two cases, depending on the last recursive rule that was applied to construct $x$.

**Case 1:** Rule 1 is the last rule applied to construct $x$. Then $x = (u)$, where $u$ is properly nested. We assume that $\text{left}[u] = \text{right}[u]$ and prove that $\text{left}[x] = \text{right}[x]$:

- $\text{left}[x] = \text{left}(u)$  
- $\text{left}(u) = 1 + \text{left}[u]$ (because $x = (u)$)
- $\text{left}[u] = \text{left}[u]$ (one more ”(“ than $u$)
- $\text{left}[x] = \text{right}[u]$ (one more “)” than $u$)
- $\text{left}[x] = \text{right}[x]$  

**Case 2:** Rule 2 is the last rule applied to construct $x$. Then $x = uv$, where $u$ and $v$ are properly nested. We assume that $\text{left}[u] = \text{right}[u]$ and $\text{left}[v] = \text{right}[v]$ and then prove that $\text{left}[x] = \text{right}[x]$:

- $\text{left}[x] = \text{left}[u] + \text{left}[v]$  
- $\text{left}[x] = \text{left}[u] + \text{left}[v]$ (because $x = uv$)
- $\text{left}[u] = \text{right}[u]$ (one more “)” than $u$)
- $\text{left}[v] = \text{right}[v]$ (one more “)” than $v$)
- $\text{left}[x] = \text{right}[x]$  

because $x = uv$
Theorem: Let \( T \) be a perfect binary tree. Then the number of vertices in \( T \) is \( 2^k - 1 \) for some positive integer \( k \).

**Proof.**

By induction.

**Base case:** the tree with one vertex has \( 2^1 - 1 = 1 \) leaves.

**Inductive step:** Let \( T' \) be a perfect binary tree. The last recursive rule that is applied to create \( T' \) takes a perfect binary tree \( T \), duplicates \( T \) and adds a new vertex \( v \) with edges to each of the roots of the two copies of \( T \). We assume that \( v(T) = 2^k - 1 \), for some positive integer \( k \) and prove that \( v(T') = 2^j - 1 \) for some positive integer \( j \).

The number of vertices in \( T' \) is twice the number of vertices in \( T \) (because of the two copies of \( T \)) plus 1 (because of the vertex \( v \) that is added), so \( v(T') = 2v(T) + 1 \). By the inductive hypothesis, \( v(T) = 2^k - 1 \) for some positive integer \( k \). Therefore

\[
v(T') = 2v(T) + 1 = 2(2^k - 1) + 1 = 2^{k+1} - 1
\]