Structural Induction

The set of natural numbers $\mathbb{N}$ has a particular structure that allows us to define it using the following recursive definition:

- $0 \in \mathbb{N}$
- if $n \in \mathbb{N}$, then $n + 1 \in \mathbb{N}$
- $\mathbb{N}$ contains nothing else

Comparable structures exist in many sets and allow us to define them recursively as follows:

i Define the "smallest" or "simplest" object (or objects) in the set

ii Define the ways in which "larger" or "more complex" objects in the set can be constructed out of "smaller" or "simpler" objects in the set

Property i is called the basis (base case), and property ii is called the induction step.

Remark. *The aforementioned recursive definition uniquely defines a set if the set in the basis contains all of the "smallest" or "simplest" objects in the set. However if a subset of these objects are used to define the basis, the recursive definition specifies an infinite number of sets that satisfy the basis and the induction step.*

Reading. *Read* **principle of set definition by recursion** *and* **Theorem 4.2** *for a rigorous description of aforementioned recursive definition [pages 97–99 of textbook].*

Example 1. *The set* $S = \{n \in \mathbb{N} : \ n \geq 2\}$ *is uniquely characterized by properties:*

i $2 \in S$

ii if $n \in S$*, then* $n + 1 \in S$

iii S *contains nothing else.*

Property iii describes 2 *as* **the** *smallest/simplest element in the set and hence insures that the above-mentioned set of properties uniquely defines* S*. Without property iii,* $\mathbb{N}$ *and* $\mathbb{Z}$ *can also satisfy properties i, and ii.*

Example 2 (Inductive Definition of Binary Trees (from textbook page 105))**.** *The smallest set* T *that satisfies the following properties is an alternative characterisation of binary trees.*

i Basis: $\Lambda \cup V$ *is in* T*. The set* V *represents the set of single nodes and* Λ *represents the empty binary tree in which* $\mathbf{nodes}(\Lambda) = \mathbf{edges}(\Lambda) = \emptyset$ *and* $\mathbf{root}(\Lambda)$ *is undefined.*

ii Induction step: Let T_1 *and* T_2 *be elements of* T *such that* $\mathbf{nodes}(T_1) \cap \mathbf{nodes}(T_2) = \emptyset$*, and* $r \notin \mathbf{nodes}(T_1) \cup \mathbf{nodes}(T_2)$*. Then the ordered triple* $T = (r, T_1, T_2)$ *is also in* T*. Furthermore, we define*

$$\begin{aligned} \boldsymbol{root}(T) &= r \\ \boldsymbol{nodes}(T) &= \{r\} \cup \boldsymbol{nodes}(T_1) \cup \boldsymbol{nodes}(T_2) \\ \boldsymbol{edges}(T) &= E \cup \boldsymbol{edges}(T_1) \cup \boldsymbol{edges}(T_2) \end{aligned}$$

where E *is the set that contains* $(r, \boldsymbol{root}(T_1))$ *if* $T_1 \neq \Lambda$*,* $(r, \boldsymbol{root}(T_2))$ *if* $T_2 \neq \Lambda$*, and nothing else.*

Reading. *Read pages 107–108 from the textbook for a proof that the aforementioned defintion is indeed an alternative charcterization of binary trees.*

Once we defined sets using a recursive definition, it seems natural to prove properties of its elements by induction. In fact, principle of simple induction follows the recursive structure for $\mathbb{N}$. **Structural Induction** is a variant of induction that is well-suited to prove the existence of a property P in a recursively defined set X. A proof by structural induction proceeds in two steps:

1. *Base case (basis):* Prove that every "smallest" or "simplest" element of X , as defined in the basis of the recursive definition, satisfies P.
2. *Induction step:* Prove that each of ways of constructing "larger" or "more complex" elements out of "smaller" or "simpler" elements, as defined by induction step of recursive definition, preserves property P.

Reading. *Read the proof by simple induction in page 101 from the textbook that shows a proof by structural induction is a proof that a property holds for all objects in the recursively defined set.*

Example 3 (Proposition 4.9 in the textbook)**.** *For any binary tree T, $|\textbf{\textit{nodes}}(T)| \leq 2^{h(T)+1} - 1$ where $h(T)$ denotes the height of tree T.*

Proof. Assume $P(T) : |\textbf{nodes}(T)| \leq 2^{h(T)+1} - 1$. We prove by structural induction that $P(T)$ holds for every binary tree.
Base case: If $T = \Lambda$, by definition $|\textbf{nodes}(T)| = 0$ and $h(T) = -1$.

$$|\textbf{nodes}(T)| = 0 = 2^0 - 1 = 2^{-1+1} - 1 = 2^{h(T)+1} - 1 \tag{1}$$

If $T \in V$ (T is a tree consisting of a single node), it is easy to see that $h(T) = 0$ and $|\textbf{nodes}(T)| = 1$. Then

$$|\textbf{nodes}(T)| = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{h(T)+1} - 1 \tag{2}$$

from (1) and (2), we can conclude that $P(T)$ holds for basis.

Induction step: Let $T_1, T_2 \in T$ and suppose that $P(T_1)$, $P(T_2)$ hold. We must prove that if $\textbf{nodes}(T_1) \cap \textbf{nodes}(T_2) = \emptyset$; and $r \notin \textbf{nodes}(T_1) \cup \textbf{nodes}(T_2)$, then $P(T)$ also holds, where $T = (r, T_1, T_2)$. The fact that nodes in T_1 and T_2 are disjoint and do not contain r allows us to write

$$|\textbf{nodes}(T_1) \cup \textbf{nodes}(T_2) \cup \{r\}| = |\textbf{nodes}(T_1)| + |\textbf{nodes}(T_2)| + 1 \tag{3}$$

Now, we can write:

$$\begin{aligned}
|\textbf{nodes}(T)| &= |\textbf{nodes}(T_1)| + |\textbf{nodes}(T_2)| && \text{(by defintion of binary tree and (3))} \\
&\leq (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1) + 1 && \text{(By IH)} \\
&\leq (2^{\max(h(T_1),h(T_2))} - 1) + (2^{\max(h(T_1),h(T_2))} - 1) + 1 \\
&\leq (2^{h(T)} - 1) + (2^{h(T)} - 1) + 1 && \text{(By defintion of height)} \\
&\leq 2^{h(T)+1} - 1
\end{aligned}$$

so $P(T)$ holds in induction step.

Hence by structural induction $P(T)$ holds for all binary trees. □

Reading. *Read Section 4.3 from textbook for an alternative to the structural induction proofs.*

Algorithm Complexity

Algorithm complexity defines a measure for running time of the algorithm by counting "elementary steps" in algorithm such as

- arithmetic operations
- assignments
- array accesses
- comparisons
- return statements
- $\vdots$

However, we know from CSC165 this is more complicated than necessary. Therefore, instead of counting each elementary step it is sufficient to count each "chunk" of instructions as 1 step, where "chunk" is a sequence of instructions that always gets executed together in constant time.

Algorithm complexity is a function $T(n)$ of input size n, which is usually the number of input elements (size of array/list, number of bits in numbers).

Algorithm complexity is a worst-case scenario measure (maximum number of steps over all inputs of the same size) which is described using asymptotic notation.

Most of the time, there exists NO simple algebraic expression for $T(n)$. Hence, we try to prove bounds on $T(n)$ when we do not know exact values! The bound on $T(n)$ can be a

1. *Upper bound:* prove worst-case time $T(n) \in O(f(n))$ by showing for some real constant $c > 0$, there exists a natural number B such that the runtime $\leq cf(n)$ for *all* inputs of size n and for all $n \geq B$
2. *Lower bound:* prove worst-case time $T(n) \in \Omega(f(n))$ by showing for some real constant $c > 0$, there exists a natural number B such that runtime $\geq cf(n)$ for *some* input of size n and for all $n \geq B$
3. *Tight bound:* prove worst-case time $T(n) \in \Theta(f(n))$ by showing $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$

In the end, recall how to deal with straight line algorithms (one statement after the other), branching (if-statements), loops, method calls.