CSE 331: Software Design & Implementation

Reference Guide

A comprehensive guide of the inductive data types, mathematical definitions, and lemmas. While those that are most commonly needed for homeworks have been included in this guide, some may be missing. If you believe that we didn't include one in this guide by mistake, let us know!

Due to the comprehensive nature of this guide, it is your responsibility to ensure that the inductive data types, mathematical definitions, or lemmas you are referencing from this guide have been covered either in lecture, section, or a previous homeworks *prior* to citing it in your homework.

Inductive Data Types

$$\textbf{type}\ \mathsf{List}_A := \mathsf{nil} \mid \mathsf{cons}(\mathsf{hd} : A,\ \mathsf{tl} : \mathsf{List}_A)$$

$$\textbf{type}\ \mathsf{Tree}_A := \mathsf{empty} \mid \mathsf{node}(x : A, S : \mathsf{Tree}_A, T : \mathsf{Tree}_A)$$

Lists – Mathematical Definitions

The following mathematical definitions return information about the given list:

$$\begin{aligned}
\textbf{func}\ &\mathsf{len}(\mathsf{nil}) &&:= 0 \\
&\mathsf{len}(\mathsf{cons}(a, L)) &&:= 1 + \mathsf{len}(L) && \text{for any } a : A \text{ and } L : \mathsf{List}_A
\end{aligned}$$

$$\begin{aligned}
\textbf{func}\ &\mathsf{sum}(\mathsf{nil}) &&:= 0 \\
&\mathsf{sum}(\mathsf{cons}(a, L)) &&:= a + \mathsf{sum}(L) && \text{for any } a : A \text{ and } L : \mathsf{List}_A
\end{aligned}$$

$$\begin{aligned}
\textbf{func}\ &\mathsf{first}(\mathsf{nil}) &&:= \text{undefined} \\
&\mathsf{first}(\mathsf{cons}(a, L)) &&:= a && \text{for any } a : A \text{ and } L : \mathsf{List}_A
\end{aligned}$$

$$\begin{aligned}
\textbf{func}\ &\mathsf{last}(\mathsf{nil}) &&:= \text{undefined} \\
&\mathsf{last}(\mathsf{cons}(a, \mathsf{nil})) &&:= a && \text{for any } a : A \\
&\mathsf{last}(\mathsf{cons}(a, \mathsf{cons}(b, L))) &&:= \mathsf{last}(\mathsf{cons}(b, L)) && \text{for any } a, b : A \text{ and } L : \mathsf{List}_A
\end{aligned}$$

$$\begin{aligned}
\textbf{func}\ &\mathsf{contains}(a, \mathsf{nil}) &&:= \text{false} && \text{for any } a : \mathbb{Z} \\
&\mathsf{contains}(a, \mathsf{cons}(b, L)) &&:= (a = b) \text{ or } \mathsf{contains}(a, L) && \text{for any } a, b : \mathbb{Z} \text{ and } L : \mathsf{List}
\end{aligned}$$

The following mathematical definition concatenates two lists together:

$$\begin{aligned}
\textbf{func}\ &\mathsf{concat}(\mathsf{nil}, R) &&:= R && \text{for any } R : \mathsf{List}_A \\
&\mathsf{concat}(\mathsf{cons}(a, L), R) &&:= \mathsf{cons}(a, \mathsf{concat}(L, R)) && \text{for any } a : A \text{ and } L, R : \mathsf{List}_A
\end{aligned}$$

The following mathematical definitions return a new list that represents the given list after it has undergone an operation (e.g. reversal):

$$\begin{array}{lll}
\textbf{func}\ \text{rev}(\text{nil}) & := \text{nil} & \\
\quad \text{rev}(\text{cons}(a, L)) & := \text{concat}(L, \text{cons}(a, \text{nil})) & \text{for any } a : A \text{ and } L : \text{List}_A
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{echo}(\text{nil}) & := \text{nil} & \\
\quad \text{echo}(\text{cons}(a, L)) & := \text{cons}(a, \text{cons}(a, \text{echo}(L))) & \text{for any } a : A \text{ and } L : \text{List}_A
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{swap}(\text{nil}) & := \text{nil} & \\
\quad \text{swap}(\text{cons}(a, \text{nil})) & := \text{cons}(a, \text{nil}) & \text{for any } a : A \\
\quad \text{swap}(\text{cons}(a, \text{cons}(b, L))) & := \text{cons}(b, \text{cons}(a, \text{swap}(L))) & \text{for any } a, b : A \text{ and } L : \text{List}_A
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{twice}(\text{nil}) & := \text{nil} & \\
\quad \text{twice}(\text{cons}(a, L)) & := \text{cons}(2a, \text{twice}(L)) & \text{for any } a : \mathbb{Z} \text{ and } L : \text{List}_{\mathbb{Z}}
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{twice-evens}(\text{nil}) & := \text{nil} & \\
\quad \text{twice-evens}(\text{cons}(a, \text{nil})) & := \text{cons}(2a, \text{nil}) & \text{for any } a : \mathbb{Z} \\
\quad \text{twice-evens}(\text{cons}(a, \text{cons}(b, L))) & := \text{cons}(2a, \text{cons}(b, \text{twice-evens}(L))) & \text{for any } a, b : \mathbb{Z} \text{ and } L : \text{List}_{\mathbb{Z}}
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{twice-odds}(\text{nil}) & := \text{nil} & \\
\quad \text{twice-odds}(\text{cons}(a, \text{nil})) & := \text{cons}(a, \text{nil}) & \text{for any } a : \mathbb{Z} \\
\quad \text{twice-odds}(\text{cons}(a, \text{cons}(b, L))) & := \text{cons}(a, \text{cons}(2b, \text{twice-odds}(L))) & \text{for any } a, b : \mathbb{Z} \text{ and } L : \text{List}_{\mathbb{Z}}
\end{array}$$

The following mathematical definitions perform operations on association lists (using the list inductive data type to implement maps)

$$\begin{array}{llll}
\textbf{func}\ \text{contains-key}(x, \text{nil}) & := \text{false} & & \text{for any } x : \mathbb{S}^* \\
\quad \text{contains-key}(x, \text{cons}((y, v), L)) & := \text{true} & \textbf{if } x = y & \text{for any } x, y, v : \mathbb{S}^* \text{ and } L : \text{List} \\
\quad \text{contains-key}(x, \text{cons}((y, v), L)) & := \text{contains-key}(x, L) & \textbf{if } x \neq y & \text{for any } x, y, v : \mathbb{S}^* \text{ and } L : \text{List}
\end{array}$$

$$\begin{array}{llll}
\textbf{func}\ \text{get-value}(x, \text{nil}) & := \text{undefined} & & \text{for any } x : \mathbb{S}^* \\
\quad \text{get-value}(x, \text{cons}((y, v), L)) & := v & \textbf{if } x = y & \text{for any } x, y, v : \mathbb{S}^* \text{ and } L : \text{List} \\
\quad \text{get-value}(x, \text{cons}((y, v), L)) & := \text{get-value}(x, L) & \textbf{if } x \neq y & \text{for any } x, y, v : \mathbb{S}^* \text{ and } L : \text{List}
\end{array}$$

The following mathematical definitions are accessory versions of the function(s) above:

$$\begin{array}{lll}
\textbf{func}\ \text{rev-acc}(\text{nil}, R) & := R & \text{for any } R : \text{List}_A \\
\quad \text{rev-acc}(\text{cons}(a, L), R) & := \text{rev-acc}(L, \text{cons}(a, R)) & \text{for any } a : A \text{ and } L, R : \text{List}_A
\end{array}$$

$$\begin{array}{lll}
\textbf{func}\ \text{sum-acc}(nil, acc) & := acc & \text{for any } acc : \mathbb{Z} \\
\quad \text{sum-acc}(\text{cons}(x, L)) & := \text{sum-acc}(L, x + acc) & \text{for any } x, acc : \mathbb{Z} \text{ and } L : \text{List}_{\mathbb{Z}}
\end{array}$$

Trees – Mathematical Definitions

The following mathematical definition returns information about the given tree:

$$
\begin{aligned}
\textbf{func}\ \mathsf{height}(\mathsf{empty}) &:= -1 \\
\mathsf{height}(\mathsf{node}(x, S, T)) &:= 1 + \max(\mathsf{height}(S), \mathsf{height}(T)) \quad \text{for any } x : A \text{ and } S, T : \mathsf{Tree}_A
\end{aligned}
$$

The following mathematical definition converts a tree to a list:

$$
\begin{aligned}
\textbf{func}\ \mathsf{toList}(\mathsf{empty}) &:= \mathsf{nil} \\
\mathsf{toList}(\mathsf{node}(x, S, T)) &:= \mathsf{concat}(\mathsf{toList}(S), \mathsf{cons}(x, \mathsf{toList}(T))) \quad \text{for any } x : \mathbb{Z} \text{ and } S, T : \mathsf{Tree}_Z
\end{aligned}
$$

Lemmas

To cite a lemma in your own proof, reference it as "Lemma #" where # is the number of the Lemma given above. The following lemmas have been proven during this class:

Lemma 1 : rev-acc(S, R) $=$ concat(rev$(S), R)$ for any S, R : List_A

Lemma 2 : concat(L, nil) $=$ L for any L : List_A

Lemma 3 : rev(rev(L)) $=$ L for any L : List_A

Lemma 4 : concat(concat$(L, R), S)$ $=$ concat$(L$, concat$(R, S))$ for any L, R, S : List_A

Arrays

Since arrays already exist in math, there is no need to redefine the type. The same array literal syntax in typescript is used in math notation, so we can write array values in math like this:

$$A := [1, 2, 3] \quad \text{(with } A : \text{Array}_{\mathbb{Z}}\text{)}$$

We write the empty array as "[]"

Properties of Arrays

Array Concatenation (#)

$$\begin{aligned} A +\!\!\!+ [\,] = A = [\,] +\!\!\!+ A & \quad \text{("identity")} \\ A +\!\!\!+ (B +\!\!\!+ C) = (A +\!\!\!+ B) +\!\!\!+ C & \quad \text{("associativity")} \end{aligned}$$

For any arrays A, B, C[1]

Subarrays

$A[i..j] = [\ A[i], A[i+1], \ldots, A[j]\ ]$ Note that this subarray includes $A[j]$

We formally define subarrays as:

$$\begin{aligned} \textbf{func } A[i..j] &:= [\,] && \text{if } j < i \\ A[i..j] &:= A[i..j-1] +\!\!\!+ [A[j]] && \text{if } 0 \le i \le j < A.length \\ A[i..j] &:= \text{undefined} && \text{if } 0 \le i \le j < A.length \end{aligned}$$

Useful facts about subarrays[1]

- $A = A[0..n-1] \qquad (= [\ A[0], A[1], \ldots, A[n-1]\ ])$
  the subarray from 0 to n-1 is the entire array (where n = A.length)
- $A[i..j] = A[i..k] +\!\!\!+ A[k+1..j]$
  holds for any $i, j, k : \mathbb{N}$ satisfying $i - 1 \le k \le j$ (and $0 \le i \le j < n$)

Sorted Arrays

$$A[j-1] \le A[j] \ \text{ for any } 1 \le j < n \quad \text{(where } n := \text{A.length)}$$

[1]These facts can be used without explanation or citation

Functions

$$
\begin{aligned}
\textbf{func}\ \mathsf{contains}([], x) &:= \mathsf{false} && && \text{for any } x : \mathbb{A} \\
\mathsf{contains}(A +\!\!+ [y], x) &:= \mathsf{true} && \textbf{if } x = y && \text{for any } x, y : \mathbb{A},\ A : \mathsf{Array}_A \\
\mathsf{contains}(A +\!\!+ [y], x) &:= \mathsf{contains}(A, x) && \textbf{if } x \neq y && \text{for any } x, y : \mathbb{A},\ A : \mathsf{Array}_A
\end{aligned}
$$

$$
\begin{aligned}
\textbf{func}\ \mathsf{sum}([]) &:= 0 \\
\mathsf{sum}(A +\!\!+ [y]) &:= \mathsf{sum}(A) + y && \text{for any } y : \mathbb{Z},\ A : \mathsf{Array}_{\mathbb{Z}}
\end{aligned}
$$