Name: ____

Algorithms and Data Structures
Jacobs University Bremen
Dr. Florian Rabe

Quiz 6
given: 2017-05-04

You have 20 minutes.

Problem 1

Points: 2+3+5

Consider the following recursive function:

$$
\begin{array}{l}
\textbf{fun } \mathit{additiveFold}(x : \mathit{IndList}[\mathbb{Z}]) : \mathbb{Z} \;= \\
\quad \textbf{match } x \\
\qquad \mathit{Nil} \mapsto 0 \\
\qquad \mathit{Cons}(\mathit{hd}, \mathit{tl}) \mapsto \mathit{hd} \;+\; \mathit{additiveFold}(\mathit{tl})
\end{array}
$$

1. Give the result of $\mathit{additiveFold}([1, 2, 3, 4])$.
2. Explain why this function is not tail-recursive.
3. Convert it to a tail-recursive function by completing the dotted parts below.

$$
\begin{array}{l}
\textbf{fun } \mathit{additiveFoldAux}(x : \mathit{IndList}[\mathbb{Z}], \mathit{result} : \mathbb{Z}) : \mathbb{Z} \;= \\
\quad \textbf{match } x \\
\qquad \mathit{Nil} \mapsto \ldots\ldots \\
\qquad \mathit{Cons}(\mathit{hd}, \mathit{tl}) \mapsto \ldots\ldots \\
\textbf{fun } \mathit{additiveFold}(x : \mathit{IndList}[\mathbb{Z}]) : \mathbb{Z} \;= \{\mathit{additiveFoldAux}(\ldots\ldots \;, \;\ldots\ldots)\}
\end{array}
$$

Solution:

1. 10
2. It does something else after receiving the result of the recursive call (in this case: the addition)
3.

$$
\begin{array}{l}
\textbf{fun } \mathit{additiveFoldAux}(x : \mathit{IndList}[\mathbb{Z}], \mathit{result} : \mathbb{Z}) : \mathbb{Z} \;= \\
\quad \textbf{match } x \\
\qquad \mathit{Nil} \mapsto \mathit{result} \\
\qquad \mathit{Cons}(\mathit{hd}, \mathit{tl}) \mapsto \mathit{additiveFold}(\mathit{tl}, \mathit{hd} + \mathit{result}) \\
\textbf{fun } \mathit{additiveFold}(x : \mathit{IndList}[\mathbb{Z}]) : \mathbb{Z} \;= \{\mathit{additiveFoldAux}(x, 0)\}
\end{array}
$$

Problem 2

Points: 5+3+2

Assume an unlabeled directed graph G with distinguished nodes *start* and *end*.

$$
\begin{array}{l}
\textbf{fun } \mathit{search}(\mathit{state} : \mathit{List}[\mathit{Node}]) : \mathit{Option}[\mathit{List}[\mathit{Node}]] \;= \\
\\
\quad \textbf{if } (\mathit{abort}(\mathit{state}))\ \{\textbf{return } \mathit{None}\} \\
\quad \textbf{if } (\mathit{solution}(\mathit{state}))\ \{\textbf{return } \mathit{Some}(\mathit{state})\} \\
\quad \mathit{foreach}(\mathit{choices}(\mathit{state}), c \mapsto \\
\qquad x := \mathit{search}(\mathit{state} + [c]) \\
\qquad \textbf{if } (x \neq \mathit{None})\ \{\textbf{return } x\} \\
\quad) \\
\quad \textbf{return } \mathit{None} \\
\\
\textbf{fun } \mathit{choices}(\mathit{state} : \mathit{List}[\mathit{Node}]) : \mathit{List}[\mathit{Node}] \;= \\
\quad \mathit{outgoing}(G, \mathit{last}(\mathit{state})) \\
\\
\textbf{fun } \mathit{solution}(\mathit{state} : \mathit{List}[\mathit{Node}]) : \mathit{bool} \;= \\
\quad \mathit{last}(\mathit{state}) == \mathit{end}
\end{array}
$$

Name: ______________________________

the last element of x

Consider the backtracking algorithm on the left.

1. What does $search([start])$ return?
2. What is the purpose of the function *abort* in general?
3. What is the purpose of the function *abort* in this particular case?

Solution:

1. a path from *start* to *end*
2. speed up the search by avoiding the search of subtrees where no solution can be found
3. abort if a cycle is found (in this case, the algorithm would keep running along the circle, i.e., search an infinite subtree; thus, aborting is not only necessary for efficiency but also to find a path at all)

Note that this algorithm is the basic algorithm for finding the exit of a maze (if intersections are represented as nodes of a graph).