Solution Sheet 3
Exercise 4
Algebraic specification of queues

module Queue;
  import Bool, true, false from Bool;
  Nat, zero from Nat;
export all;
sort Queue;
operations
  newqueue : -> Queue;
  enqueue : Queue x Nat -> Queue;
  isempty : Queue -> Bool;
  dequeue : Queue -> Queue;
  head : Queue -> Nat;
  tail: Queue -> Nat;
declare q : Queue; n,m : Nat;
axioms
  isempty(newqueue) == true;
  isempty(enqueue(q,n)) == false;
  dequeue(newqueue) == newqueue;
  dequeue(enqueue(newqueue,n)) == newqueue;
  dequeue(enqueue(enqueue(q,m),n)) == enqueue(dequeue(enqueue(q,m)),n);
  tail(newqueue) == zero;
  tail(enqueue(q,n)) == n;
  head(newqueue) == zero;
  head(enqueue(newqueue,n)) == n;
  head(enqueue(enqueue(q,m),n)) == head(enqueue(q,m));
end module Queue;
Decomposition into constructors and other operations

module Queue;

import Bool, true, false from Bool;
Nat, zero from Nat;
export all;
sort Queue;

constructors
newqueue : -> Queue;
enqueue : Queue x Nat -> Queue;

operations
isempty : Queue -> Bool;
dequeue : Queue -> Queue;
head : Queue -> Nat;
tail: Queue -> Nat;

operation axioms
isempty(newqueue) == true;
isempty(enqueue(q,n)) == false;
dequeue(newqueue) == newqueue;
dequeue(enqueue(newqueue,n)) == newqueue;
dequeue(enqueue(enqueue(q,m),n)) == enqueue(dequeue(enqueue(q,m)),n);
tail(newqueue) == zero;
tail(enqueue(q,n)) == n;
head(newqueue) == zero;
head(enqueue(newqueue,n)) == n;
head(enqueue(enqueue(q,m),n)) == head(enqueue(q,m));

end module Queue;
Constraints for constructive specifications

- The outermost operation of a left-hand side of an axiom is no constructor, all inner operations are constructors.
- A variable occurs at most once on the left-hand side.
- All variables of the right-hand side occur on the left-hand side.
- The system of axioms is **unique** with respect to a (non-constructor) operation, i.e. for each tuple of argument terms there is at most one matching rule.
- The system of axioms is **complete** with respect to a (non-constructor) operation, i.e. for each tuple of argument terms there is at least one matching rule.
- The system of axioms is **terminating**, i.e. for variable-free terms there are only derivations of finite length.
Representation of queues

Informal representation

\[
q = \text{enqueue}(\text{enqueue}(\text{enqueue}(\text{enqueue}(\text{enqueue}(\text{newqueue}, 5), 4), 3), 2), 1)
\]

Term

Tree
Execution of head(q)

head(q) ==
head(enqueue(enqueue(enqueue(enqueue(enqueue(newqueue,5),4),3),2),1)) ==
head(enqueue(enqueue(enqueue(enqueue(newqueue,5),4),3),2)) ==
head(enqueue(enqueue(enqueue(newqueue,5),4),3)) ==
head(enqueue(enqueue(newqueue,5),4)) ==
head(enqueue(newqueue,5)) ==
5

Execution of head(q) requires linear time and should require constant time!
Exercise 5
module ArrayNatNat;

(* Record composed of an array and two natural numbers. *)
import Array from Array; Nat from Nat;
export all;
sort ArrayNatNat;
constructor (_,_,_) : Array x Nat x Nat -> ArrayNatNat;
operations
arrayOf _ : ArrayNatNat -> Array;
nat1Of _ : ArrayNatNat -> Nat;
nat2Of _ : ArrayNatNat -> Nat;
_[/array] : ArrayNatNat x Array -> ArrayNatNat;
_[/nat1] : ArrayNatNat x Nat -> ArrayNatNat;
_[/nat2] : ArrayNatNat x Nat -> ArrayNatNat;
declare a : Array; n1, n2 : Nat; ann : ArrayNatNat;
operation axioms
    arrayOf((a,n1,n2)) == a;
    nat1Of((a,n1,n2)) == n1;
    nat2Of((a,n1,n2)) == n2;
    ann[a/array] == (a, nat1Of ann, nat2Of ann);
    ann[n1/nat1] == (arrayOf ann, n1, nat2Of ann);
    ann[n2/nat2] == (arrayOf ann, nat1Of ann, n2);
end module ArrayNat;
Implementation of queues

module \texttt{Queue};
\begin{verbatim}
import \texttt{ArrayNatNat, (\_,\_,\_), arrayOf \_, nat1Of \_, nat2Of \_ from ArrayNatNat;}
\end{verbatim}
\begin{verbatim}
Array, empty, \_[:,:], read from \texttt{Array}; \texttt{Bool from Bool;}
\end{verbatim}
\begin{verbatim}
Nat, zero, succ, pre, \_ = \_, \_ < \_ from \texttt{Nat}
\end{verbatim}
\begin{verbatim}
rename \texttt{Nat as Item, zero as \texttt{0, zero as error, succ as \_+1, pre as \_-1;}
\end{verbatim}
\begin{verbatim}
export all;
\end{verbatim}
\begin{verbatim}
operations
\end{verbatim}
\begin{verbatim}
\texttt{newqueue} : \rightarrow \texttt{ArrayNatNat};
\texttt{enqueue} : \texttt{ArrayNatNat} x \texttt{Item} \rightarrow \texttt{ArrayNatNat};
\texttt{isempty} : \texttt{ArrayNatNat} \rightarrow \texttt{Bool};
\texttt{dequeue} : \texttt{ArrayNatNat} \rightarrow \texttt{ArrayNat} ;
\texttt{head} : \texttt{ArrayNatNat} \rightarrow \texttt{Item};
\texttt{tail} : \texttt{ArrayNatNat} \rightarrow \texttt{Item};
\end{verbatim}
\begin{verbatim}
declare \texttt{ann} : \texttt{ArrayNat}; \texttt{it}, \texttt{it1}, \texttt{it2} : \texttt{Item}; \texttt{a} : \texttt{Array};
\end{verbatim}
\begin{verbatim}
axioms
\texttt{newqueue} == (\texttt{empty}, \texttt{0, 0});
\texttt{enqueue}(\texttt{ann},\texttt{it}) ==
\begin{verbatim}
(arrayOf \texttt{ann}[\texttt{it}/\texttt{nat2Of \texttt{ann}}], \texttt{nat1Of \texttt{ann}}, \texttt{nat2Of \texttt{ann} + 1});
\end{verbatim}
\texttt{isempty}((\texttt{a}, \texttt{n1}, \texttt{n2})) == \texttt{n1} = \texttt{n2};
\texttt{dequeue}((\texttt{a}, \texttt{n1}, \texttt{n2}) ==
\begin{verbatim}
if \texttt{n1=\texttt{n2}} then (\texttt{a, n1, n2}) else (\texttt{a, n1+1, n2});
\end{verbatim}
\texttt{head}((\texttt{a,n1,n2})) == if \texttt{n1} = \texttt{n2} then \texttt{error} else \texttt{read(a,n1)};
\texttt{tail}((\texttt{a,n1,n2}) == if \texttt{n2} = \texttt{n1} then \texttt{error} else \texttt{read(a,n2-1)};
\end{verbatim}
end module \texttt{Queue};
Data representations

- Assuming a built-in array type, each queue would be represented by an array \( a \) with minimal index \( n_1 \) and maximal index \( n_2 - 1 \).
- All operations require constant time.
- head accesses the element with index \( n_1 \).