spec Queues

declares

section GraphSchema

declares

node class QUEUE
  intrinsic
   Name : string;
end;

node type Queue : QUEUE end;

node class ELEMENT
  intrinsic
   Data : integer := 0;
end;

node type Element : ELEMENT end;

class Head : QUEUE [0:1] -> ELEMENT [1:1];
class Tail : QUEUE [0:1] -> ELEMENT [1:1];
class Next : ELEMENT [0:1] -> ELEMENT [0:1];
end;
section ExportedOperations

declares

production newqueue( name : string ; out queue : Queue) =

[`
[``]
```
::=

```
1' : Queue
```
```
<table>
<thead>
<tr>
<th>Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
</tr>
</tbody>
</table>
```
```
2' : Element
```
```
3' : Element
```
```
transfer 1'.Name := name;
return queue := 1';
end;

(*An empty queue is represented with the help of two dummy elements, whose data attributes are never accessed. This representation was chosen such that each exported operation may be encoded by just one corresponding graph test or graph production without having to use negative application conditions, etc. *)
test isempty( queue : Queue) =

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
  \node (queue) {\texttt{'1 = queue}};
  \node (element1) [below left of=queue] {\texttt{'2 : Element}};
  \node (element2) [below right of=queue] {\texttt{'3 : Element}};
  \node (element3) [below of=element2] {\texttt{'3 : Element}};

  \draw[->] (queue) to node {Tail} (element1);
  \draw[->] (queue) to node {Head} (element2);
  \draw[->] (element1) to node {Next} (element2);
\end{tikzpicture}
\end{center}

end;
production enqueue( queue : Queue ; data : integer ; out element : ELEMENT) =

\[ '1 = \text{queue} \]

\[ \text{Tail} \]

\[ '2 : \text{Element} \rightarrow '3 : \text{Element} \]

\[ '2' = '1 \]

\[ \text{Tail} \]

\[ 2' = '2 \rightarrow 4' : \text{Element} \rightarrow 3' = '3 \]

\[ \text{transfer} \ 4'.Data := data; \]
\[ \text{return} \ element := 4'; \]
\[ \text{end}; \]

(*By the representation chosen, it is guaranteed that there is an outgoing Next edge from the dummy tail element. The new element is inserted into the chain of Next edges. Note that this production exploits the fact that partial graphs rather than subgraphs are required in PROGRES: A Head edge from ‘1 to ‘3 may or may not be present. *)
production dequeue( queue : Queue) =

::=

end;
(*Inverse operation to enqueue, see also comment above. *)
test head(queue : QUEUE; out data : integer) =

\[\begin{array}{c}
\text{\texttt{Head}} \\
\text{\texttt{\textasciitilde 1 = queue}} \\
\text{\texttt{\texttt{\textasciitilde 3 : Element}} \rightarrow \texttt{\texttt{\textasciitilde 2 : Element}}} \\
\end{array}\]

\[\text{return data := \texttt{\textasciitilde 3.Data;}}\]

end;
test tail( queue : QUEUE ; out data : integer) =

\[ \text{\texttt{\textbackslash '1} = \text{queue} } \]

\[ \text{\texttt{\textbackslash '1}} \text{\rightarrow} \text{\texttt{\textbackslash '3} : \text{Element} } \]

\[ \text{\texttt{\textbackslash '3}} \text{\rightarrow} \text{\texttt{\textbackslash '2} : \text{Element} } \]

\[ \text{return data := \texttt{\textbackslash '2.Data; } \]

end;

end;

end;

end.