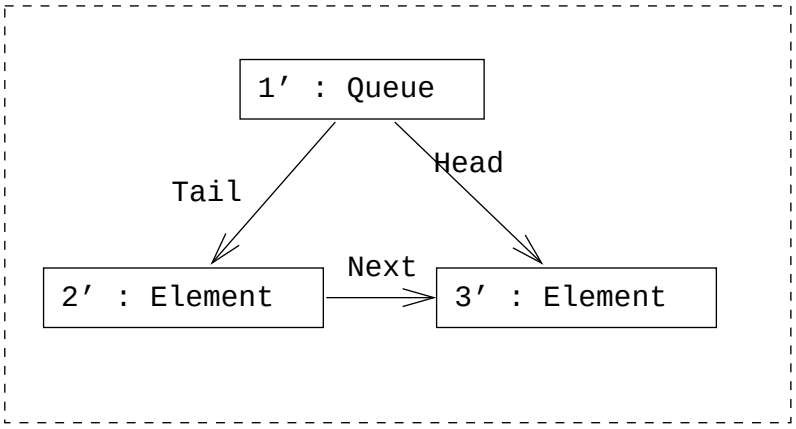
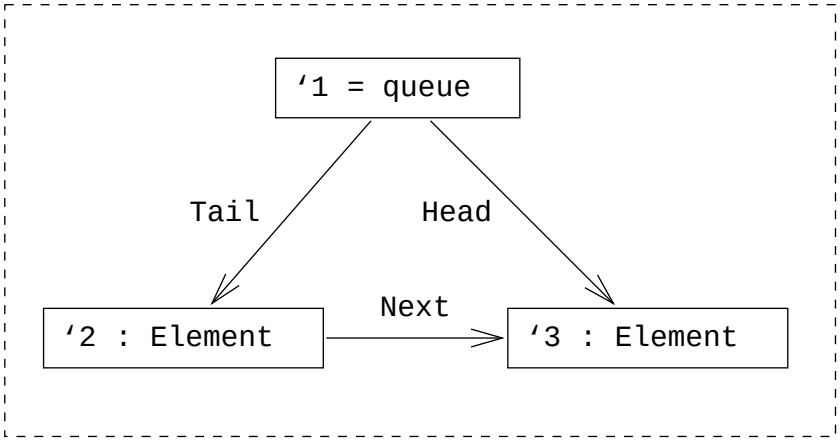
```
spec Queues

  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;

        edge type Head : QUEUE [0:1] -> ELEMENT [1:1];

        edge type Tail : QUEUE [0:1] -> ELEMENT [1:1];

        edge type Next : ELEMENT [0:1] -> ELEMENT [0:1];

    end;
```

```
section ExportedOperations

  declares

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

    [ ]

      ::=
```



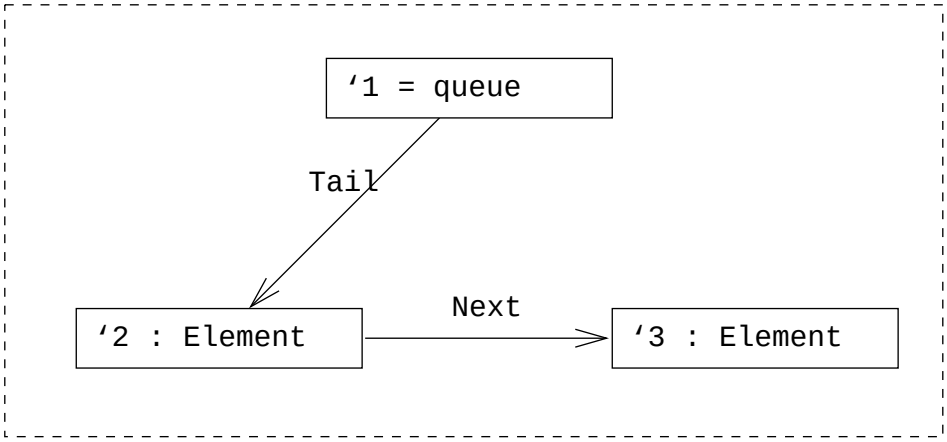

```
      transfer 1’.Name := name;
      return queue := 1’;
    end;
    (*An empty queue is represented with the help of two dummy
    elements, whose data attributes are never accessed. This
    representation was chosen such that each exported operation
    may be encoded by just one corresponding graph test or
    graph production without having to use negative application
    conditions, etc.    *)
```

```
test isempty( queue : Queue) =
```

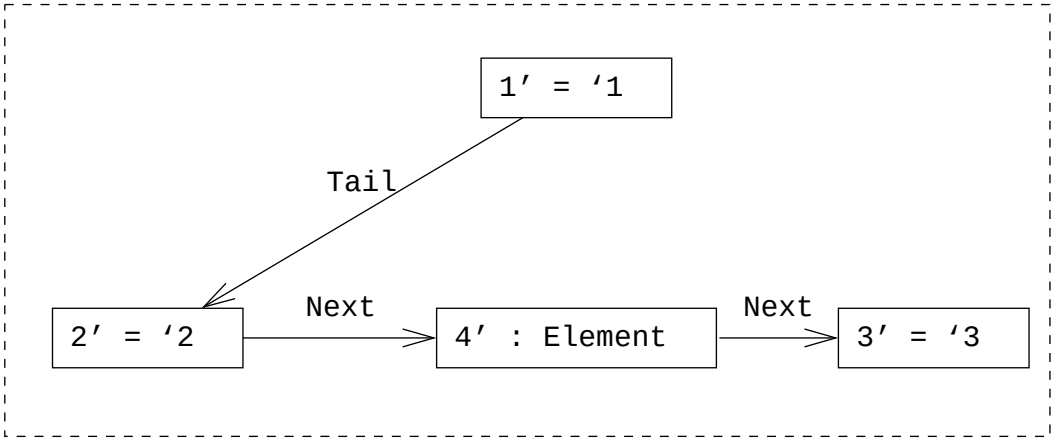



```
end;
```

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

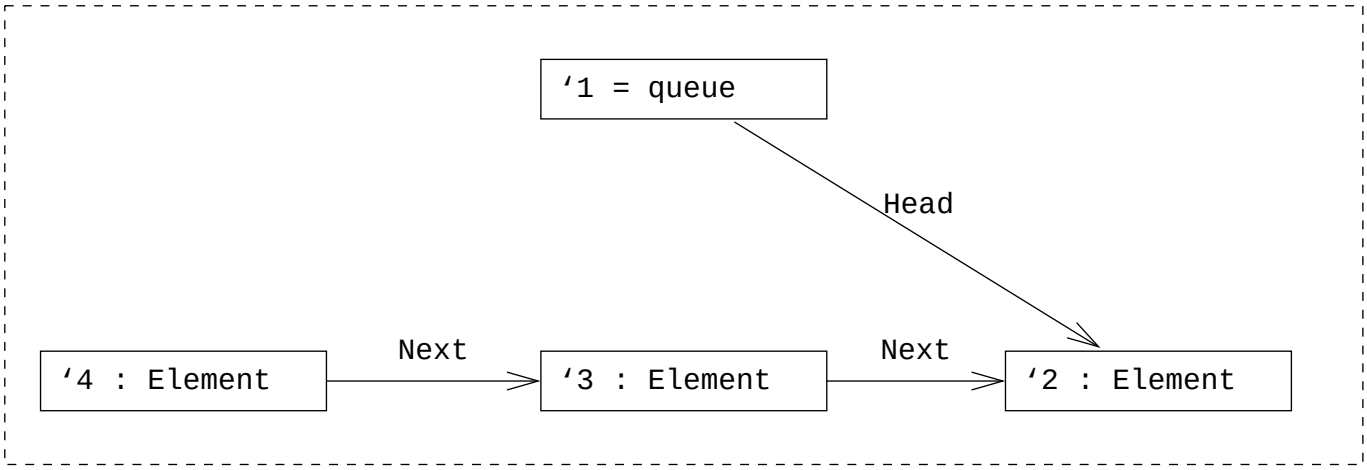


::=

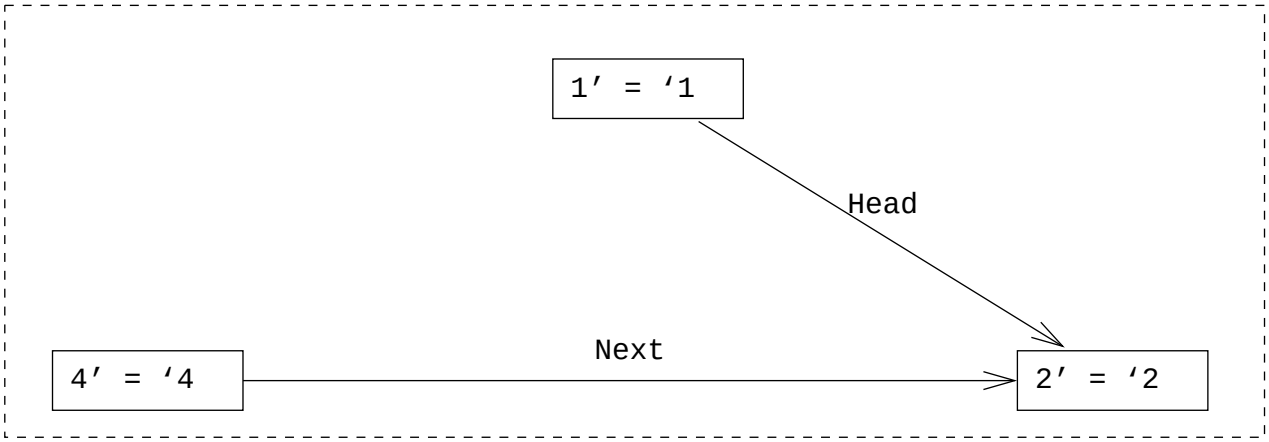



```
  transfer 4’.Data := data;
  return element := 4’;
end;
(*By the representation chosen, it is guaranteed that there is an
outgoing Next edge from the dummy tail element. The new element
is inserted into the chain of Next edges. Note that this production
exploits the fact that partial graphs rather than subgraphs
are required in PROGRES: A Head edge from ’1 to ’3 may or may
not be present.    *)
```

production dequeue(queue : Queue) =

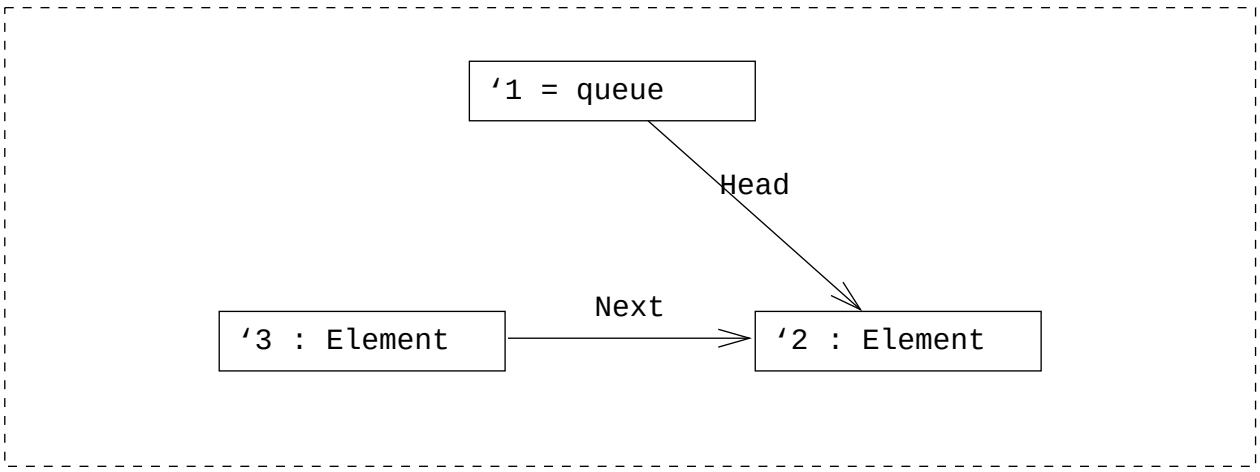


::=



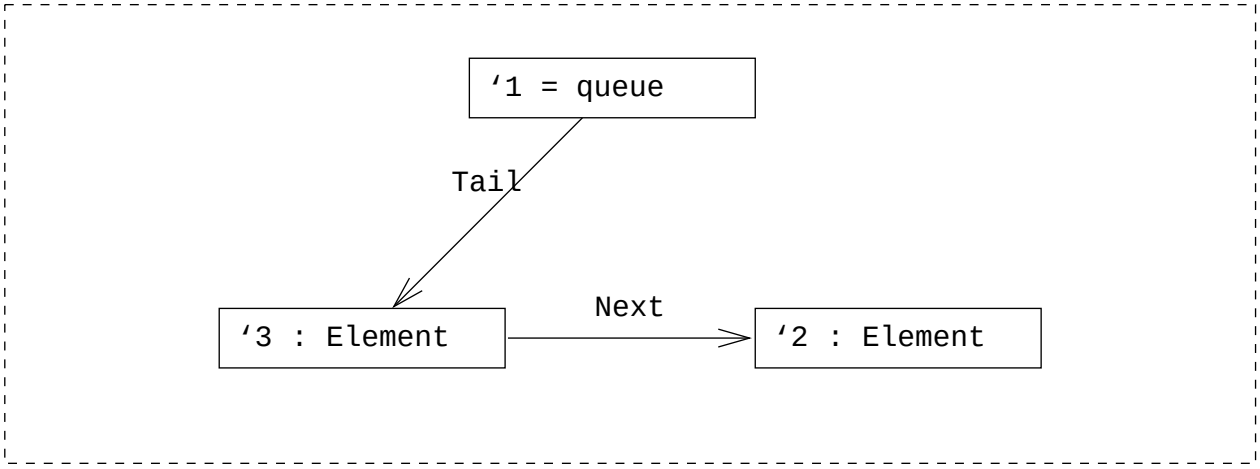
end;
(*Inverse operation to enqueue, see also comment above. *)

```
test head( queue : QUEUE ; out data : integer) =
```




```
  return data := ‘3.Data;

end;
```

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




```
      return data := ‘2.Data;
    end;
  end;
end.
```