Fundamentals of Numerical Analysis and Symbolic Computation
Exercise (June 4)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

April 28, 2009

The result is to be submitted by the deadline stated above via the Moodle interface of the course "Fundamentals of Numerical Analysis and Symbolic Computation" as a PDF file (with an appropriate cover page and a section for each exercise, hand-written results are also acceptable but have to be scanned and included in the file).

Exercise 1: Queues

Specify an abstract datatype with sort *Queue* whose values represent *queues* of elements from sort *Elem*. The datatype shall support at least the following operations:

- *empty* ... the empty queue.
- *enqueue*(e, q) ... the queue constructed from queue q by appending element e to the tail of q.
- *front*(q) ... the element at the front of q.
- *dequeue*(q) ... the queue constructed from q by removing the front element.
- *length*(q) ... the number of elements in q.

A queue obeys the FIFO ("first in/first out") principle i.e. the first element enqueued is also the first element dequeued.

Write a loose specification with (possibly free) constructors and use in your formulations of the axioms (possibly conditional) equational logic. You may leave the sort *Elem* unspecified.

Write a second specification of a sort *PQueue* whose elements represent *priority queues* of elements.. A priority queue differs from an ordinary queue in that *front*(q) does not return the element at the front of q but the smallest element of q (according to a relation $\_ \leq \_ : \mathit{Elem} \times \mathit{Elem} \rightarrow \mathit{Bool}$ which is to be specified as a total order).

Exercise 2: Graphs

Specify an abstract datatype with sort *Graph* whose values represent finite directed graphs. The datatype shall support at least the following operations:

- *empty* ... the empty graph,
- *vertex*(v, g) ... the graph constructed from graph g by adding vertex v.
- *arc*(v_1, v_2, g) ... the graph constructed from graph g by adding an arc from vertex v_1 to vertex v_2.
- *isvertex*(v, g) ... true iff v is a vertex in g.
- *isarc*(v_1, v_2, g) ... true iff there is an arc from v_1 to v_2 in g.
- *vertices*(g) ... the set of vertices of g.
- *incoming*(v, g) ... the set of vertices in g such that there exists an arc from each of these vertices to v.
- *indegree*(v, g) the number of arcs in g with target v.
- *outgoing*(v, g) ... the set of vertices in g such that there exists an arc from v to each of these vertices.
- *outdegree*(v, g) ... the number of arcs in g with source g.
- *maxindeg*(g) ... the maximum number of arcs leading to a vertex in g.
- *maxoutdeg*(g) ... the maximum number of arcs leaving a vertex in g.
- *remove*(v_1, v_2, g) ... the graph constructed from g by removing the arc from v_1 to v_2.
- *remove*(v, g) ... the graph constructed from g by removing vertex v (and all arcs from/to v).
- *order*(g) the number of vertices in g.
- *size*(g) the number of arcs in g.
- *connected*(v_1, v_2, g) ... true iff there exists a path (a possibly empty sequence of edges) from v_1 to v_2 in g.
- *distance*(v_1, v_2, g) ... the length of the shortest path from v_1 to v_2 in g.

You may leave the sort *Vertex* of vertices unspecified and assume the usual set operations as predefined.

Write a loose specification with (possibly free) constructors and use in your formulations of the axioms full predicate logic with equality.

Do not overconstrain your specifications by defining concrete operation results for "erroneous" inputs; if your operations only make sense under certain assumptions, make these explicit as antecedents of implications, e.g.

$$\forall v{:}\mathit{Vertex}, g{:}\mathit{Graph}\ .\ \mathit{isvertex}(v, g) \Rightarrow \mathit{indegree}(v, g) = \ldots$$