Formal Specification of Abstract Datatypes
Exercise 1 (May 4)

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

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- a PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the formal specifications in the style of "Thinking Programs",
  - the CafeOBJ specifications,
  - comprehensive tests of the CafeOBJ specifications (sample reductions),
  - optionally any explanations or comments you would like to make;
- the CafeOBJ (.mod) file(s) of the specifications.

Exercise: Queues

A *queue* is a "First In/First Out" data structure to which elements can be added at the tail and from which elements can be removed at the head[1].

Formal Specification First, develop a formal specification of the data type `Queue` in the style presented in the course:

```
spec QUEUE[sort Elem] import NAT :=
free type Queue = empty | enqueue(Queue,Elem)
then {
  fun head: Queue → Elem
  fun dequeue: Queue → Queue
  fun size: Queue → Nat
  pred has ⊆ Queue × Elem
  ...
}
```

with the following intended interpretations:

- `empty` denotes the queue without any elements;
- `enqueue` adds an element to the tail of the queue;
- `head` denotes the element at the head of the queue;
- `dequeue` removes an element from the head of the queue;
- `size` denotes the number of elements in the queue;
- `has` tells whether the queue holds a specific element.

Here `Queue` is freely generated by constructors `empty` and `enqueue`, i.e., every queue can be uniquely represented by a term of form

```
enqueue(enqueue(...enqueue(empty,e_1)...,e_(n-1)),e_n)
```

You may assume that `NAT` provides the usual operations on natural numbers (which may be written in the usual infix style).

Is the abstract datatype denoted by your specification monomorphic or not? Justify your answer.

[1] http://en.wikipedia.org/wiki/Queue_(data_structure)

CafeOBJ Second, implement the specification in CafeOBJ by a tight module:

```
module! QUEUE
{
  protecting (NAT)
  signature
  {
    [ Queue ]
    op empty : -> Queue
    op enqueue : Queue Nat -> Queue
    ...
  }
  axioms
  {
    ...
  }
}
```

Here you may for simplicity replace sort `Elem` by sort `Nat`.

Test the CafeOBJ module with several reductions. Give the input and output of each test and your interpretation of the results (do they indicate errors in your specification or not?). If your specification contains errors, use the trace facilities of CafeOBJ for debugging.