Exceptions — A Counterexample

The specification

In this “example”, we re-consider the Queue ADT. To illustrate the potential pitfalls in treating exceptions, we present a mistaken attempt to include the treatment of errors/exceptions in its description. For brevity, we use the signature

NEW: Queue
ADD: Queue, Int $\rightarrow$ Queue
DEL: Queue $\rightarrow$ Queue $\cup$ $\{ERROR_Q\}$
FRT: Queue $\rightarrow$ Int $\cup$ $\{ERROR_I\}$

The situations we wish to account for are when there is an attempt to access the front of an empty Queue, or delete the first item of an empty Queue. Neither of these operations is possible and we wish to specify an error in these cases. To this end, we have added idealized error elements into the appropriate domains in the signature, and propose the equations below.

For all $q \in$ Queue, and $i,j \in$ Int
1. FRT(NEW) = $ERROR_I$
2. FRT(ADD(NEW),i))= i
3. FRT(ADD(ADD(q,i),j)) = FRT(ADD(q,i))
4. DEL(NEW) = $ERROR_Q$
5. DEL(ADD(NEW,i)) = NEW
6. DEL(ADD(ADD(q,i),j)) = ADD(DEL(ADD(q,i)),j)

Introducing these idealized error elements into the domains necessitates specifying the behavior of the functions when such a value is an argument. We elect the “errors propagate” paradigm. While this is often left to be implicitly understood, to be on firm ground we need to be able to describe it in the same way other behavior is described. So here we specify this through the equations below.

7. FRT($ERROR_Q$) = $ERROR_I$
8. DEL($ERROR_Q$) = $ERROR_Q$
9. ADD(q,$ERROR_I$) = $ERROR_Q$
10. ADD($ERROR_Q$,i) = $ERROR_Q$

Note that behaviors other than “errors propagate” may be desirable. An instance is equation 9, where adding one error element leads to the destruction of an entire queue of well-formed values.

The mistake
While each of the equations in this specification appears reasonable in isolation, together, there is a catastrophic blunder! Consider the queue

q = DEL(ADD(NEW, FRT(NEW))).

By equation 5, q ≡ NEW. However, by equation 1, q ≡ DEL(ADD(NEW, ERROR_I)), and by equations 8 and 9 DEL(ADD(NEW, ERROR_I)) ≡ DEL(ERROR_Q) ≡ ERROR_Q. Therefore NEW ≡ ERROR_Q!! And then by equations 9 and 10, *every* queue is equivalent to ERROR_Q!

Note that while these equivalencies are a horrible blunder and completely invalidate the specification, this does not demonstrate an *inconsistency* since the unintended collapsing occurs in the TOI, not in a pre-defined type. However, this specification is also, in fact, inconsistent. For example consider the term
i = FRT(ADD(ADD(NEW,2), FRT(NEW)).
By equations 2 and 3, i ≡ FRT(ADD(NEW,2)) ≡ 2. But by equations 1, 7 and 9, i ≡ FRT(ADD(ADD(NEW,2), ERROR_I)) ≡ FRT(ERROR_Q) ≡ ERROR_I. Hence ERROR_I ≡ 2, an inconsistency.

One apparent solution to such problems is to condition “non-error” equations. For example, consider

3**'**. FRT(ADD(ADD(q,i),j)) = **if** j=ERROR_I **then** ERROR_I **else** FRT(ADD(q,i))

However, this is not legal — only a valid Boolean expression is allowed as the condition of an if-then-else (recall its definition)! However, this is the germ of the idea that we do develop to reliably treat errors.