Algebraic Specification of Abstract Data Types in Z

Bart Massey
2016-05-24

ADTs

An *Abstract Data Type* is a poorly-defined and old concept in SE. A standard way to formalize it in the modern world is with an *algebraic description*.

An algebra consists of sets (sorts, types) of objects (carrier sets), a set of operations (functions) closed over those objects (all operations take arguments of carrier set type and return a result of carrier set type), and a collection of laws (equations) that constrain how the operations work.

For example:

$$Counter = \{\mathbb{Z};\ incr, decr\}$$

$$\begin{aligned}
&new : Counter \\
&val : Counter \rightarrow \mathbb{Z} \\
&incr : Counter \rightarrow Counter \\
&decr : Counter \rightarrow Counter
\end{aligned}$$

$$\begin{aligned}
&val(new) = 0 \\
&\forall x : Counter \bullet incr(decr(x) = decr(incr(x)) = x \\
&\forall x : Counter \bullet val(incr(x)) > val(x) \\
&\forall x : Counter \bullet val(decr(x)) < val(x)
\end{aligned}$$

Z ADTs

An ADT specification looks an awful lot like a Z specification. The most problematic is the actual definition of *Counter*, which looks like it should just be a schema.

$$
\begin{array}{|l}
\hline \textit{Counter} \\
\quad new : Counter \\
\quad val : Counter \rightarrow \mathbb{Z} \\
\quad incr : Counter \rightarrow Counter \\
\quad decr : Counter \rightarrow Counter \\
\hline
\quad val(new) = 0 \\
\quad \forall x : Counter \bullet incr(decr(x)) = decr(incr(x)) = x \\
\quad \forall x : Counter \bullet val(incr(x)) > val(x) \\
\quad \forall x : Counter \bullet val(decr(x)) < val(x) \\
\hline
\end{array}
$$

Unfortunately, this won't typecheck, since we cannot use *Counter* until it is defined. Probably better, if a little odd, is to make *Counter* just be a type and use a generic schema to capture its operations and laws.

$$
\begin{array}{|l}
\hline \textit{CounterADT}[Counter] \\
\quad new : Counter \\
\quad val : Counter \rightarrow \mathbb{Z} \\
\quad incr : Counter \rightarrow Counter \\
\quad decr : Counter \rightarrow Counter \\
\hline
\quad val(new) = 0 \\
\quad \forall x : Counter \bullet incr(decr(x)) = decr(incr(x)) = x \\
\quad \forall x : Counter \bullet val(incr(x)) > val(x) \\
\quad \forall x : Counter \bullet val(decr(x)) < val(x) \\
\hline
\end{array}
$$

Note the key ADT property: we cannot tell anything about the structure of *Counter* except what is implied by the laws.

ADT Implementation

Let us start with the obvious implementation of *CounterADT*.

$$
\begin{array}{|l}
\hline \textit{CounterZ} \\
\quad CounterADT[\mathbb{Z}] \\
\hline
\quad new = 0 \\
\quad \forall x : \mathbb{Z} \bullet val(x) = x \\
\quad \forall x : \mathbb{Z} \bullet incr(x) = x + 1 \\
\quad \forall x : \mathbb{Z} \bullet decr(x) = x - 1 \\
\hline
\end{array}
$$

The key question is whether this implementation obeys the laws of counters:

$$
\begin{array}{l}
val(new) = val(0) = 0 \\
\forall\, x : Counter \bullet \\
\quad incr(decr(x)) = (x - 1) + 1 = x \\
\quad decr(incr(x)) = (x + 1) - 1 = x \\[1ex]
\forall\, x : Counter \bullet val(incr(x)) = val(x + 1) = x + 1 \ > \\
\quad val(x) = x \\
\forall\, x : Counter \bullet val(decr(x)) = val(x - 1) = x - 1 \ < \\
\quad val(x) = x
\end{array}
$$

So…yes.

Partial Operations Are Awkward

One nice property that *Counter* has as specified is that every operation is a total function. It is not uncommon, though, that we would prefer a "natural" counter such that

$$\forall\, x : NatCounter \bullet val(x) \geq 0$$

We could try just doing the obvious thing and adding this law to the counter laws.

$$
\begin{array}{|l}
\hline
NatCounterADTUnsound[NatCounter] \\
\hline
new : NatCounter \\
val : NatCounter \to \mathbb{Z} \\
incr : NatCounter \to NatCounter \\
decr : NatCounter \to NatCounter \\
\hline
\forall\, x : NatCounter \bullet val(x) \geq 0 \\[1ex]
val(new) = 0 \\
\forall\, x : NatCounter \bullet incr(decr(x)) = decr(incr(x)) = x \\
\forall\, x : NatCounter \bullet val(incr(x)) > val(x) \\
\forall\, x : NatCounter \bullet val(decr(x)) < val(x) \\
\hline
\end{array}
$$

Unfortunately, we get into trouble immediately: *decr* can no longer be a total function.

$$
\begin{array}{lr}
val(new) = 0 & [1\text{: given}] \\
\forall\, x : NatCounter \bullet val(decr(x)) < val(x) & [2\text{: given}] \\
\forall\, x : NatCounter \bullet val(x) \geq 0 & [3\text{: given}] \\
val(decr(new)) < 0 & [4\text{: (1), }\forall\text{-inst (2)}] \\
val(decr(new)) \geq 0 & [5\text{: }\forall\text{-inst (2)}] \\
\neg\, (val(decr(new)) < 0) & [6\text{: math (5)}] \\
\Box & [\Box\text{-intro (4), (6)}]
\end{array}
$$

In a way, the fact that we can prove our specification unsound is good news. This keeps us from building a program that will have runtime errors. However, we have to figure out what to do about it. There are three standard approaches.

Restrict Operation Domains

The easiest thing to do is simply to restrict the domain of $decr$.

$$
\begin{array}{|l}
\hline
\textit{NatCounterADTRestrict}[\textit{Counter}] \\
\quad \textit{CounterADT}[\textit{Counter}] \\
\hline
\quad \operatorname{ran}(val) = \mathbb{N} \\
\quad \operatorname{dom}(decr) = \textit{Counter} \setminus \{new\} \\
\hline
\end{array}
$$

This isn't quite right: we must also relax the laws of $\textit{CounterADT}$ a bit so that $incr(decr(new))$ is undefined.

Note that we now have a proof obligation every time we use $decr$: we must prove that it is not being passed new. This is probably good practice and the right way to go, but it can significantly complicate proofs.

Force Operations To Be Total

We could certainly insist that the $decr$ function always return a result with non-negative val. The obvious way to do this is to modify the laws so that decrementing from zero just returns zero again.

$$
\begin{array}{|l}
\hline
\textit{NatCounterADTTotal}[\textit{NatCounter}] \\
\quad new : \textit{NatCounter} \\
\quad val : \textit{NatCounter} \to \mathbb{Z} \\
\quad incr : \textit{NatCounter} \to \textit{NatCounter} \\
\quad decr : \textit{NatCounter} \to \textit{NatCounter} \\
\hline
\quad \forall x : \textit{NatCounter} \bullet val(x) \geq 0 \\
\\
\quad val(new) = 0 \\
\quad decr(new) = new \\
\quad \forall x : \textit{NatCounter} \bullet decr(incr(x)) = x \\
\quad \forall x : \textit{NatCounter} \mid x \neq new \bullet incr(decr(x)) = x \\
\quad \forall x : \textit{NatCounter} \bullet val(incr(x)) > val(x) \\
\quad \forall x : \textit{NatCounter} \mid x \neq new \bullet val(decr(x)) < val(x) \\
\hline
\end{array}
$$

Unfortunately, this revised counter "acts weird". Some of the laws of the unsound counter were things we wanted to hold, and now they don't. The behavior that calling *decr* may not actually decrease the counter, in particular, is surprising and will probably lead to bugs in the code that uses counters.

Lift To An Error Value

Let us define a generic type for values that either indicate an error or a non-error value.

$$[\mathit{GenericCounter}]$$
$$\mathit{RESULT} ::= \mathit{nope} \mid \mathit{ok}\langle\!\langle \mathit{GenericCounter} \rangle\!\rangle$$
$$\mathit{RESULTN} ::= \mathit{nopen} \mid \mathit{okn}\langle\!\langle \mathbb{N} \rangle\!\rangle$$

We can now rewrite the laws to have an explicit nope when decrementing too far.

NatCounterADTLifted

$$\mathit{new} : \mathit{RESULT}$$
$$\mathit{val} : \mathit{RESULT} \rightarrow \mathit{RESULTN}$$
$$\mathit{incr} : \mathit{RESULT} \rightarrow \mathit{RESULT}$$
$$\mathit{decr} : \mathit{RESULT} \rightarrow \mathit{RESULT}$$

$$\mathit{val}(\mathit{new}) = \mathit{okn}(0)$$

$$\mathit{decr}(\mathit{new}) = \mathit{nope}$$

$$\forall x : \mathit{RESULT} \bullet x = \mathit{decr}(\mathit{incr}(x))$$

$$\forall x : \mathit{RESULT} \mid x \neq \mathit{new} \bullet \mathit{incr}(\mathit{decr}(x)) = x$$

$$\mathit{val}(\mathit{nope}) = \mathit{nopen}$$

$$\forall x : \mathit{RESULT};\ y, z : \mathbb{N} \mid x \neq \mathit{nope} \land \mathit{okn}(y) = \mathit{val}(\mathit{incr}(x)) \land \mathit{okn}(z) = \mathit{val}(x) \bullet y > z$$

$$\mathit{val}(\mathit{nope}) = \mathit{nopen}$$

$$\forall x : \mathit{RESULT};\ y, z : \mathbb{N} \mid x \notin \{\mathit{nope}, \mathit{new}\} \land \mathit{okn}(y) = \mathit{val}(\mathit{decr}(x)) \land \mathit{okn}(z) = \mathit{val}(x) \bullet y < z$$

$$\mathit{val}(\mathit{decr}(\mathit{new})) = \mathit{nopen}$$

Notice that this is a huge mess, making proofs tough. It also punts all errors to runtime. Not a great choice either.