Seq Satisfies Set

Garlan & Wing Handout 11 9 October 1995

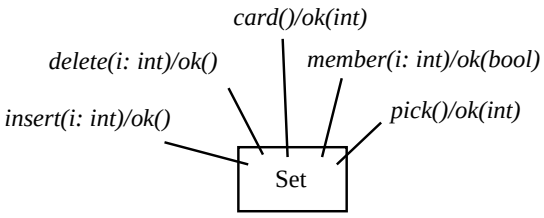
The motivation for this example is show you that when you do "object-oriented" programming, you are really identifying certain abstract objects (better known an *data abstractions* or *abstract data types*) like sets, stacks, queues, symbol tables, etc.. You eventually have to realize (i.e., implement) these objects in a real programming language in terms of either other abstract objects or the language's built-in data objects like sequences, arrays, records, linked lists, etc. After you write your (concrete) implementation you are then faced with proving it correct with respect to the (abstract) specification.

Not surprisingly, these data objects (abstract or built-in) can themselves be viewed as little state machines. So, to show the correctness of an implementation of an abstract object is very much like showing that one state machine (the concrete one) satisfies another (the abstract one).

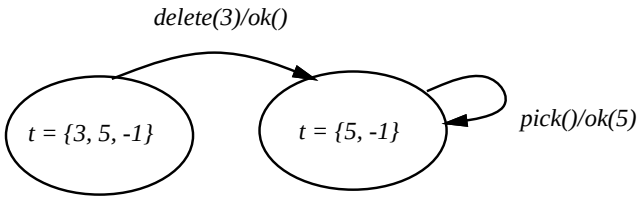
There are other proof techniques used to prove the correctness of the implementations of abstract data types. You may see them in the Methods Course (when you learn about object-oriented programming) and/or the Analysis course (when you learn about program verification).

1 Abstract Machine: Set

My Set abstract machine has the following interface:



Set's Interface



Part of Set's State Transition Diagram

Set = (
$\{s : \{t\} \rightarrow set[int]\}$,
$\{s : \{t\} \rightarrow set[int] \mid s(t) = \emptyset\}$,
$\{insert(i : int)/ok(), \ldots$ see above $\ldots, pick()/ok(int)\}$,
$\delta_A = \ldots$ see next page $\ldots$
).

Here are specifications of the actions, *insert, delete, card, member, and pick.*

©1995 Jeannette M. Wing

insert(i: int)/ok()
pre *true*
post $t' = t \cup \{i\}$

delete(i: int)/ok()
pre *true*
post $t' = t \setminus \{i\}$

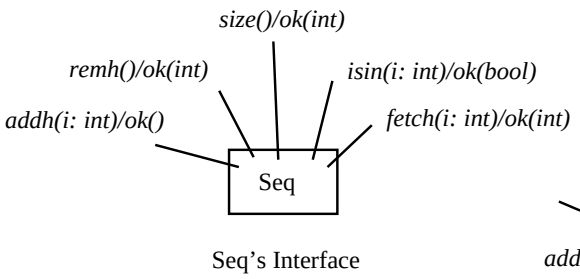
card()/ok(int)
pre *true*
post $t' = t \wedge result = \#t$

member(i: int)/ok(bool)
pre *true*
post $t' = t \wedge result = (i \in t)$

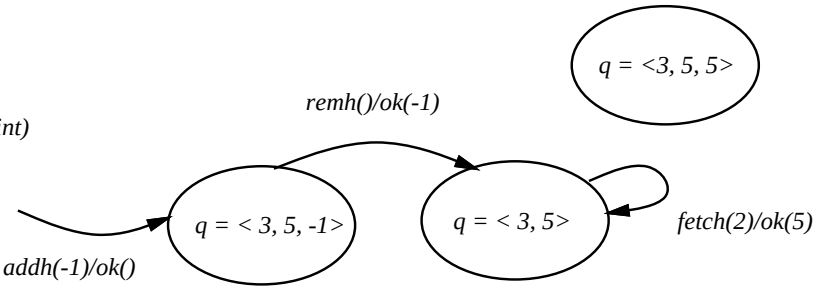
pick()/ok(int)
pre $t \neq \emptyset$
post $t' = t \wedge result \in t$

2 Concrete Machine: Seq

I've decided that I want to implement the Set machine in terms of a Seq machine with the following interface:



Seq's Interface



Part of Seq's State Transition Diagram

I'm going to define the actions such that the state where $q = \langle 3, 5, 5 \rangle$ is unreachable. You'll see why soon.

Seq = (
$\{s : \{q\} \rightarrow seq[int]\}$,
$\{s : \{q\} \rightarrow seq[int] \mid s(q) = \langle\rangle\}$,
$\{addh(i : int)/ok(), \ldots$ see above $\ldots, fetch(i : int)/ok(int)\}$,
$\delta_C = \ldots$ see next page $\ldots$
).

Seq's actions have the following specification:

addh(i: int)/ok()
pre $i \notin ran\ q$
post $q' = q \frown \langle i \rangle$

remh()/ok(int)
pre $q \neq \langle\rangle$
post $q = q' \frown \langle result \rangle$

size()/ok(int)
pre *true*
post $q' = q \wedge result = \#q$

isin(i: int)/ok(bool)
pre *true*
post $q' = q \wedge result = (i \in ran\ q)$

fetch(i: int)/ok(int)
pre $q \neq \langle\rangle \wedge i \in dom\ q$
post $q' = q \wedge result = q\ i$

3 Proof of Correctness

Formally, RI and AF are defined over Seq's set of states, but it's going to be notationally convenient (and I hope more understandable) if I denote each of these states by the sequence value to which Seq's state variable, q, maps. In other words, I should write something like:

$$RI(s(q) = \langle 3, 5, -1 \rangle) = \ldots$$

but instead I'm going to write:

$$RI(\langle 3, 5, -1 \rangle) = \ldots$$

3.1 Step 1

I first need to define an abstraction function and a representation invariant.

1. Abstraction Function.
Informally AF takes an (ordered) sequence of elements and turns it into an (unordered) set of the sequence's elements. Formally,

$$AF : seq[int] \twoheadrightarrow set[int]$$
$$AF(\langle\rangle) = \varnothing$$
$$AF(q \frown \langle e \rangle) = AF(q) \cup \{e\}$$

It is common for abstraction functions to be defined recursively like this.

Notice that this AF is many-to-one. There are many sequence values that map to the same set value because we don't care what the order of elements is in a set. In fact, the orderedness property of sequences is exactly the "irrelevant" property from which we abstract. For example,

$$AF(\langle 3, 5, -1 \rangle) = \{3, 5, -1\}$$
$$AF(\langle 5, 3, -1 \rangle) = \{3, 5, -1\}$$
$$AF(\langle -1, 5, 3 \rangle) = \{3, 5, -1\}$$

These three different sequence values map to the same set value.

2. Representation Invariant.

Notice that the *addh* action has a pre-condition that checks whether the element to be inserted is already in the sequence. Thus, only sequence values that have no duplicate elements serve to represent set values. We have the following representation invariant, which characterizes the domain of AF:

$$RI : seq[int] \rightarrow bool$$
$$RI(q) = \forall\, 1 \leq i, j \leq \#q \bullet i \neq j \Rightarrow q\, i \neq q\, j$$

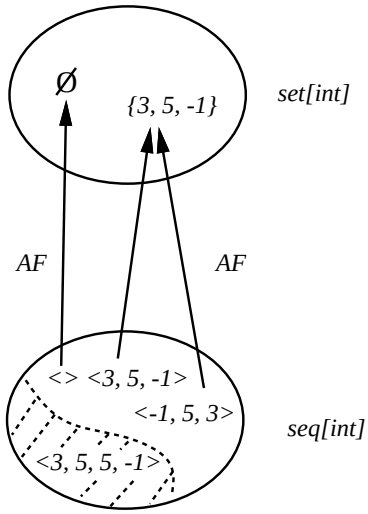
Informally I'll call this representation invariant, NoDups. Thus we have,

$RI(\langle 3, 5, 5, -1 \rangle)$	$= false$	5 appears twice.
$RI(\langle 3, 5, -1 \rangle)$	$= true$	NoDups
$RI(\langle\rangle)$	$= true$	The empty sequence is ok.

All those sequence values that RI maps to *true* are legal representations of set values.

IMPORTANT: Remember that there's a side proof that we need to do here. We need to show that the representation invariant is indeed an invariant. That is, along the lines of the proof technique described in the handout on Reasoning About State Machines, we need to show that the invariant is established in the initial states and preserved by each action. I'll leave this part of the proof as an exercise to the reader.

Here's a picture illustrating that AF is partial and many-to-one:



3.2 Step 2

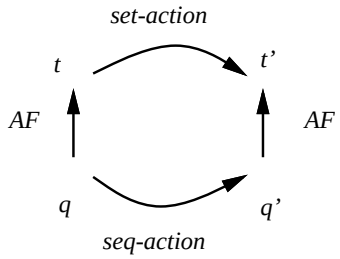
Armed with RI and AF, I can now show the concrete machine satisfies the abstract one.

1. Initial condition.

We need to show that each initial state of Seq maps to some initial state of Set. More formally, we need to show that $AF(\langle\rangle) = \varnothing$. This is obviously true by the definition of AF.

2. Commuting diagram for each Seq action.

I need to show this diagram:



$$\delta_A(AF(q), \text{set-action}) = AF(\ \delta_C(q, \text{seq-action}))$$

There are five cases, one for each Seq action. I'll just do three actions, *addh*, *remh*, and *size*.

Case: *addh* satisfies *insert*.

We'll first consider the case where i is not in the sequence and then the case for when it is.

Case 1: $i \notin ran\ q$

According to the post-condition for Set's *insert* action we need to show that the set value for t' obtained after doing an *insert* with argument i is $t \cup \{i\}$. Seq's *addh* action has the effect of adding to the high end of the sequence only if its argument i is not already stored in the sequence. Thus if q is the value of the sequence before, then $q \frown \langle i \rangle$ is the value after. In other words, we have:

$AF(\delta_C(q, addh(i)/ok()))$	
$= AF(q) \frown \langle i \rangle)$	post-condition of *addh*
$= AF(q) \cup \{i\}$	def'n of AF
$= t \cup \{i\}$	since $t = AF(q)$
$= t'$	post-condition of *insert*

Case 2: $i \in ran\ q$

If i is already in the sequence then no state transition occurs and q stays the same.

Case: *remh* satisfies *delete*.

According to the post-condition of Set's *delete* action the set value for t' obtained after doing the *delete* is the set with i removed. Seq's *remh* action has the effect of removing and returning the high end of the original sequence q. Informally speaking, it's this element *result*, which we would "pass to" *delete* as an argument. In other words, given a particular state transition involving *remh*, I am choosing a particular state transition involving *delete*—the one for which *result* is passed as an argument. We have:

$\delta_A(AF(q), delete(result)/ok())$	
$= \delta_A(AF(q' \frown \langle result \rangle), delete(result)/ok())$	post-condition of *remh*
$= \delta_A(AF(q') \cup \{result\}, delete(result)/ok())$	def'n of AF
$= (AF(q') \cup \{result\}) \setminus \{result\}$	post-condition of *delete* (and def'n of δ_A)
$= AF(q')$	properties about set union and set difference
$= t'$	$t' = AF(q')$

Case: *size* satisfies *card*

Looking at the post-condition for Set's *card* action, there are two things to show.

First, we need to show that the value returned by the Seq's *size* action is the size of the corresponding set (under AF). Because of NoDups (the RI), we know that the size of the sequence representing a set is the size of the set it represents. More formally, you would need to prove a lemma like:

Lemma 1 $\forall\, q : seq[int] \bullet (\#q = \#AF(q))$

Second, we need to show that *size* does not change the abstract value of the set that q represents. More formally,

$q' = q$	
$\Rightarrow$	post-condition of *size*
$AF(q') = AF(q)$	
$\Rightarrow$	AF is a function.
$t' = t$	

IMPORTANT: Notice that we rely on the abstraction function AF on being a function here. In the second step above, we apply AF to two equal things; since AF is a function (and not a relation), we know the result of applying AF to two equal things will result in two equal things.

For your homework, it's okay to give informal proofs like the ones I give here.