Answers to the mid-term examination on Algebraic Specification

Christian Rinderknecht

22 April 2005

Question 1. We want an algebraic specification of sets made of elements of the same type element. Let us call it SET(element). Here is the signature:

- **Defined types**
  - The type of the sets is noted t.
  - The type of the set elements is noted element.
- **Constructors**
  - EMPTY : t
    The term EMPTY represents the set (noted $\varnothing$ in mathematics) which contains no element.
  - ADD : element $\times$ t $\to$ t
    The term ADD(e, s) denotes the set s augmented by adding element e. If e was already in s, then the result is s. The order in which elements are added is not meaningful, so all function on sets should not depend on any element order.
- **Functions**
  - UNION : t $\times$ t $\to$ t
    The term UNION(s_1, s_2) denotes the set made of the elements of s_1 and s_2 (noted $s_1 \cup s_2$ in mathematics). It is commutative and associative.
  - INTER : t $\times$ t $\to$ t
    The term INTER(s_1, s_2) denotes the set made of elements of s_1 which also belong to s_2 (noted $s_1 \cap s_2$ in mathematics). The function name stands for "intersection". It is commutative and associative. It is distributive over the set union.

– MEM : element × t → boolean
The term MEM(e, s) is **true** if and only if e belongs to s, otherwise it is **false**. The function name is a shorthand for "membership".

Complete this signature with equations defining the constructors and functions.

Answer 1. There are two constructors for sets. Since one (EMPTY) models the empty set, the other one (ADD) necessarily models the non empty sets.

Usually, there is no need for equations for the constructors but, here, it is said that "If e was already in s, then the result is s", so some equations are required to formalise this property (it is a constraint).

Two cases can occur when we define a non empty set ADD(e, s): e is already in s or it is missing. In fact, we only need to take care of the first case, since if e is absent in s, we just let ADD(e, s) as it is.

But how do we find out whether e is in s or not? If s is empty, we do not need an equation: ADD(e, EMPTY) remains itself. So, let us consider the case where s is not empty. Then it must be constructed with ADD, so let $s = \text{ADD}(e_1, s_1)$. In other words, we have to find an equation like

$$\text{ADD}(e, \text{ADD}(e_1, s_1)) = \textbf{?}$$

Two cases occur: if $e = e_1$ or not:

$$\text{ADD}(e, \text{ADD}(e, s_1)) = \textbf{?} \tag{1}$$
$$\text{ADD}(e, \text{ADD}(e_1, s_1)) = \textbf{?} \qquad \text{where } e \neq e_1 \tag{2}$$

Equation 1 is easy to complete: the right-hand side is simply s. So

$$\begin{aligned} \text{ADD}(e, \text{ADD}(e, s_1)) &= \text{ADD}(e, s_1) \\ \text{ADD}(e, \text{ADD}(e_1, s_1)) &= \textbf{?} \qquad \text{where } e \neq e_1 \end{aligned}$$

In equation 2, the left-hand side is equal to the set made of adding e to s_1 and then e_1 (because we know $e \neq e_1$). Formally:

$$\begin{aligned} \text{ADD}(e, \text{ADD}(e, s_1)) &= \text{ADD}(e, s_1) \\ \text{ADD}(e, \text{ADD}(e_1, s_1)) &= \text{ADD}(e_1, \text{ADD}(e, s_1)) \qquad \text{where } e \neq e_1 \end{aligned}$$

Let us try on some examples on integer sets. Let

$$s = \text{ADD}(3, \text{ADD}(1, \text{ADD}(7, \text{EMPTY})))$$

First let us add 1 to s:

$$\begin{aligned}
\textsc{Add}(1, s) &= \textsc{Add}(\mathbf{1}, \textsc{Add}(\mathbf{3}, \textsc{Add}(1, \textsc{Add}(7, \textsc{Empty})))) \\
&= \textsc{Add}(3, \textsc{Add}(\mathbf{1}, \textsc{Add}(\mathbf{1}, \textsc{Add}(7, \textsc{Empty})))) && \text{because } 1 \neq 3 \\
&= \textsc{Add}(3, \textsc{Add}(1, \textsc{Add}(7, \textsc{Empty})))) && \text{because } 1 = 1
\end{aligned}$$

Now let us add 4 to s:

$$\begin{aligned}
\textsc{Add}(4, s) &= \textsc{Add}(\mathbf{4}, \textsc{Add}(\mathbf{3}, \textsc{Add}(1, \textsc{Add}(7, \textsc{Empty})))) \\
&= \textsc{Add}(3, \textsc{Add}(\mathbf{4}, \textsc{Add}(\mathbf{1}, \textsc{Add}(7, \textsc{Empty})))) && \text{because } 4 \neq 3 \\
&= \textsc{Add}(3, \textsc{Add}(1, \textsc{Add}(\mathbf{4}, \textsc{Add}(\mathbf{7}, \textsc{Empty})))) && \text{because } 4 \neq 1 \\
&= \textsc{Add}(3, \textsc{Add}(1, \textsc{Add}(7, \textsc{Add}(4, \textsc{Empty})))) && \text{because } 4 \neq 7
\end{aligned}$$

Now let us consider the defining equations for function Union, which models the union of sets. The signature tells us that this function takes two arguments of type `t`, i.e. two sets. Since a set can be empty or non empty, this means that we have to consider a maximum of four cases:

$$\textsc{Union}(\textsc{Empty}, \textsc{Empty}) = \mathbf{?} \tag{3}$$
$$\textsc{Union}(\textsc{Empty}, \textsc{Add}(e, s)) = \mathbf{?} \tag{4}$$
$$\textsc{Union}(\textsc{Add}(e, s), \textsc{Empty}) = \mathbf{?} \tag{5}$$
$$\textsc{Union}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) = \mathbf{?} \tag{6}$$

The left-hand side of equation 3 corresponds to the mathematical expression $\varnothing \cup \varnothing$. So it is easy to find the right-hand side:

$$\begin{aligned}
\textsc{Union}(\textsc{Empty}, \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Union}(\textsc{Empty}, \textsc{Add}(e, s)) &= \mathbf{?} \\
\textsc{Union}(\textsc{Add}(e, s), \textsc{Empty}) &= \mathbf{?} \\
\textsc{Union}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) &= \mathbf{?}
\end{aligned}$$

The left-hand side of equation 4 corresponds to $\varnothing \cup x$, where $x \neq \varnothing$, and in equation 5 to $x \cup \varnothing$, with $x \neq \varnothing$. So, again, the right-hand sides are simple to find:

$$\begin{aligned}
\textsc{Union}(\textsc{Empty}, \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Union}(\textsc{Empty}, \textsc{Add}(e, s)) &= \textsc{Add}(e, s) \\
\textsc{Union}(\textsc{Add}(e, s), \textsc{Empty}) &= \textsc{Add}(e, s) \\
\textsc{Union}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) &= \mathbf{?}
\end{aligned}$$

In equation 6, we have two elements, e_1 and e_2, which must be in the union, as well as the union of s_1 and s_2. So, finally we have:

$$\begin{aligned}
\textsc{Union}(\textsc{Empty}, \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Union}(\textsc{Empty}, \textsc{Add}(e, s)) &= \textsc{Add}(e, s) \\
\textsc{Union}(\textsc{Add}(e, s), \textsc{Empty}) &= \textsc{Add}(e, s) \\
\textsc{Union}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) &= \textsc{Add}(e_1, \textsc{Add}(e_2, \textsc{Union}(s_1, s_2)))
\end{aligned}$$

Note that equations 1 and 2 of function Add take care of ignoring possible redundant elements, for example if e_1 is actually equal to e_2.

It is possible to simplify the previous equations into

$$\begin{aligned}
\textsc{Union}(\textsc{Empty}, s) &= s \\
\textsc{Union}(\textsc{Add}(e_1, s_1), s) &= \textsc{Add}(e_1, \textsc{Union}(s, s_1))
\end{aligned}$$

Why do we write $\textsc{Union}(s, s_1)$ instead of $\textsc{Union}(s_1, s)$? Actually, both are correct because the order of the elements in a set is not meaningful. But if we want the *same* order as in equations 3 to 6, we have to write $\textsc{Union}(s, s_1)$. Consider the simplified equations where $s = \textsc{Add}(e_2, s_2)$:

$$\begin{aligned}
\textsc{Union}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) &= \textsc{Add}(e_1, \textsc{Union}(\textsc{Add}(e_2, s_2), s_1)) \\
&= \textsc{Add}(e_1, \textsc{Add}(e_2, \textsc{Union}(s_1, s_2)))
\end{aligned}$$

Also we should not forget that the signature says that Union is commutative ($x \cup y = y \cup x$) and associative ($x \cup (y \cup z) = (x \cup y) \cup z$), hence we must add the corresponding equations:

$$\textsc{Union}(s_1, s_2) = \textsc{Union}(s_2, s_1) \tag{7}$$
$$\textsc{Union}(s_1, \textsc{Union}(s_2, s_3)) = \textsc{Union}(\textsc{Union}(s_1, s_2), s_3) \tag{8}$$

Now let us consider the equations of the set intersection Inter.

The signature, as for the union, informs us that this function takes two sets. So we have four cases at the most:

$$\textsc{Inter}(\textsc{Empty}, \textsc{Empty}) = \textbf{?} \tag{9}$$
$$\textsc{Inter}(\textsc{Empty}, \textsc{Add}(e, s)) = \textbf{?} \tag{10}$$
$$\textsc{Inter}(\textsc{Add}(e, s), \textsc{Empty}) = \textbf{?} \tag{11}$$
$$\textsc{Inter}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) = \textbf{?} \tag{12}$$

Left-hand sides of equations 9, 10 and 11 correspond respectively to $\varnothing \cap \varnothing$, $\varnothing \cap x$ and $x \cap \varnothing$, where $x \neq \varnothing$. Therefore the right-hand sides are easy to

guess:

$$\begin{aligned}
\textsc{Inter}(\textsc{Empty}, \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Empty}, \textsc{Add}(e, s)) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Add}(e, s), \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Add}(e_1, s_1), \textsc{Add}(e_2, s_2)) &= \mathbf{?}
\end{aligned}$$

In the last equation, we keep e_1 in the intersection if and only if it belongs to $\textsc{Add}(e_2, s_2)$. This can be checked using the function Mem. Otherwise, e_1 must be skipped. If we let $s = \textsc{Add}(e_2, s_2)$ we get

$$\begin{aligned}
\textsc{Inter}(\textsc{Empty}, \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Empty}, \textsc{Add}(e, s)) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Add}(e, s), \textsc{Empty}) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Add}(e_1, s_1), s) &= \textsc{Add}(e_1, \textsc{Inter}(s_1, s)) && \text{if } \textsc{Mem}(e_1, s) \\
\textsc{Inter}(\textsc{Add}(e_1, s_1), s) &= \textsc{Inter}(s_1, s) && \text{if not } \textsc{Mem}(e_1, s)
\end{aligned}$$

We can even simplify the equations if we note that in the two last ones, only the first set decreases: it will eventually becomes Empty after repeatingly applying these equations from left to right. Then equation 10 will apply if the second set is not Empty. In order to cope with this case, we can simply change equation 10 by allowing Empty as second argument. So we just can keep the new equation 10 and the two last ones:

$$\begin{aligned}
\textsc{Inter}(\textsc{Empty}, s) &= \textsc{Empty} \\
\textsc{Inter}(\textsc{Add}(e_1, s_1), s) &= \textsc{Add}(e_1, \textsc{Inter}(s_1, s)) && \text{if } \textsc{Mem}(e_1, s) \\
\textsc{Inter}(\textsc{Add}(e_1, s_1), s) &= \textsc{Inter}(s_1, s) && \text{if not } \textsc{Mem}(e_1, s)
\end{aligned}$$

By removing equation 11, we make the conclusion longer to reach if the second set is empty, but the result is the same (this is the same phenomenon that happens with the Append function on stacks). The signature says also that the intersection is commutative, associative, as well as distributive over the set union. This means in mathematics: $x \cap y = y \cap x$, $x \cap (y \cap z) = (x \cap y) \cap z$ and $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$. In terms of our specification these equations become:

$$\textsc{Inter}(s_1, s_2) = \textsc{Inter}(s_2, s_1) \tag{13}$$

$$\textsc{Inter}(s_1, \textsc{Inter}(s_2, s_3)) = \textsc{Inter}(\textsc{Inter}(s_1, s_2), s_3) \tag{14}$$

$$\textsc{Inter}(s_1, \textsc{Union}(s_2, s_3)) = \textsc{Union}(\textsc{Inter}(s_1, s_2), \textsc{Inter}(s_1, s_3)) \tag{15}$$

Finally, let us consider the defining equations of function MEM, which models the set membership. The signature tells us that this function takes two arguments: the first one is a set element and the second is a set.

As usual, let us consider two kinds of sets: empty and non empty (this is because we only have two constructors):

$$\text{MEM}(e, \text{EMPTY}) = \textbf{?} \tag{16}$$
$$\text{MEM}(e, \text{ADD}(e_1, s_1)) = \textbf{?} \tag{17}$$

If we check for the membership of any element in the empty set, the value is **false** (the return type of the function is boolean):

$$\begin{aligned} \text{MEM}(e, \text{EMPTY}) &= \textbf{false} \\ \text{MEM}(e, \text{ADD}(e_1, s_1)) &= \textbf{?} \end{aligned}$$

Now two new situations are possible: either $e = e_1$ or $e \neq e_1$. In the first case, let $e = e_1 = e_2$. In other words:

$$\begin{aligned} \text{MEM}(e, \text{EMPTY}) &= \textbf{false} \\ \text{MEM}(e, \text{ADD}(e, s_1)) &= \textbf{?} \\ \text{MEM}(e, \text{ADD}(e_1, s_1)) &= \textbf{?} \qquad\qquad \text{where } e \neq e_1 \end{aligned}$$

The first case is actually straightforward: we found e belonging to $s = \text{ADD}(e, s_1)$:

$$\begin{aligned} \text{MEM}(e, \text{EMPTY}) &= \textbf{false} \\ \text{MEM}(e, \text{ADD}(e, s_1)) &= \textbf{true} \\ \text{MEM}(e, \text{ADD}(e_1, s_1)) &= \textbf{?} \qquad\qquad \text{where } e \neq e_1 \end{aligned}$$

In the last case, e may belong to s_1, we have to check that:

$$\begin{aligned} \text{MEM}(e, \text{EMPTY}) &= \textbf{false} \\ \text{MEM}(e, \text{ADD}(e, s_1)) &= \textbf{true} \\ \text{MEM}(e, \text{ADD}(e_1, s_1)) &= \text{MEM}(e, s_1) \qquad\qquad \text{where } e \neq e_1 \end{aligned}$$

Question 2. Consider the STACK(item) signature as follows:

- **Defined types**
  - The type of the stacks is called t.
  - The type of the items is called item.

- **Constructors**
  - EMPTY : t
    Term EMPTY denotes the empty stack.
  - PUSH : item × t → t
    Term PUSH(e, s) denotes stack s with item e on the top.
- **Functions**
  - POP : t → item × t
    This is the projection of PUSH.
  - APPEND : t × t → t
    Term APPEND(s_1, s_2) denotes the stack made of s_2 on the bottom and s_1 on the top.
  - NTH : t × int → item
    Term NTH(s, n) denotes the n^{th} item in stack s. The item must exist.
  - FLATTEN : STACK(STACK(item).t).t → STACK(item)
    Term FLATTEN(s), where s denotes a stack of stack of items, represents the stack made by appending all the stacks in s, in the same order.
  - EXISTS : (item → boolean) × STACK(item).t → boolean
    Term EXISTS(f, s) is **true** if and only if there exists an item e in s such that $f(e)$ is **true**. Otherwise it is **false**.

Give equations defining NTH, FLATTEN and EXISTS.

Answer 2. First let us consider the equations defining function NTH. The signature tells us that this function takes two arguments: the first one is a stack and the second one is an integer corresponding to the position in the stack.

We understand that the item at the top of the stack, if any, is the first one. This means that NTH$(s, 0)$ has no meaning, hence no equation about it (the equations define, that is they are the meaning of a function).

The signature says also that the sought item must exist in the stack, so NTH(EMPTY, n) for any n, where EMPTY denotes the empty stack, has no meaning, thus no equation.

What are the remaining cases? We still have to consider non empty stacks, i.e. of the kind PUSH(e, s), and positions greater than 1, distinguishing the

case 1:

$$\textsc{Nth}(\textsc{Push}(e, s), 1) = \textbf{?} \tag{18}$$
$$\textsc{Nth}(\textsc{Push}(e, s), n) = \textbf{?} \qquad \text{where } n > 1 \tag{19}$$

Equation 18 is already answered: the top item on the stack is the fist one. So

$$\textsc{Nth}(\textsc{Push}(e, s), 1) = e$$
$$\textsc{Nth}(\textsc{Push}(e, s), n) = \textbf{?} \qquad \text{where } n > 1$$

In order to complete equation 19, a good picture is very useful. Since $n > 1$, the item we look for is not e. So we must look in s. But the top item of s is the *second* of $\textsc{Push}(e, s)$, not the first one. So the item we look for is at position $n - 1$ in s, whilst being at position n in $\textsc{Push}(e, s)$. Therefore

$$\textsc{Nth}(\textsc{Push}(e, s), 1) = e$$
$$\textsc{Nth}(\textsc{Push}(e, s), n) = \textsc{Nth}(s, n - 1) \qquad \text{where } n > 1$$

Let us consider the equations for equations defining function Flatten. Here again, a good picture is very helpful. The signatures informs us that this function takes only one argument which is a stack of stack of some items. It is convenient to imagine a non empty stack as a stack of boxes. In this case, these boxes themselves contain stacks of smaller boxes. The effect of Flatten is to take all these smaller stacks and to append all of them in a single stack.

Since stacks can only be constructed using two constructors, let us consider two equations, one about empty stacks and the other about non-empty ones:

$$\textsc{Flatten}(\textsc{Empty}) = \textbf{?} \tag{20}$$
$$\textsc{Flatten}(\textsc{Push}(e, s)) = \textbf{?} \tag{21}$$

Equation 20 is simple: if there no stack then there are no items at all.

$$\textsc{Flatten}(\textsc{Empty}) = \textsc{Empty}$$
$$\textsc{Flatten}(\textsc{Push}(e, s)) = \textbf{?}$$

In equation 21, item e is a stack also (the smaller stacked boxes) and we have to put it on top of the flattening of s. This is achieved by use of Append:

$$\textsc{Flatten}(\textsc{Empty}) = \textsc{Empty}$$
$$\textsc{Flatten}(\textsc{Push}(e, s)) = \textsc{Append}(e, \textsc{Flatten}(s))$$

Finally let us consider the equations defining function EXISTS. The signature says that this function takes two arguments: the first one is a predicate over the type item (item $\rightarrow$ boolean) and the second one is a stack containing items of type item (STACK(item).t).

The purpose of the function call EXISTS(f, s) is to tell if there exists an item in the stack s which satisfy a given property f, or not.

The first step is to deconstruct a general stack into two kinds: empty and non-empty:

$$\text{EXISTS}(f, \text{EMPTY}) = \textbf{?} \tag{22}$$
$$\text{EXISTS}(f, \text{PUSH}(e, s)) = \textbf{?} \tag{23}$$

Equation 22 is straightforward to complete: by construction, there is no item in the empty stack:

$$\begin{aligned} \text{EXISTS}(f, \text{EMPTY}) &= \textbf{false} \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{?} \end{aligned}$$

Now, we have to consider two more cases: either $f(e)$ (is **true**) or $\neg f(e)$ (is **true**)[1]:

$$\begin{aligned} \text{EXISTS}(f, \text{EMPTY}) &= \textbf{false} & \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{?} & \text{if } f(e) \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{?} & \text{if } \neg f(e) \end{aligned}$$

If $f(e)$ then we found the kind of item we were looking for:

$$\begin{aligned} \text{EXISTS}(f, \text{EMPTY}) &= \textbf{false} & \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{true} & \text{if } f(e) \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{?} & \text{if } \neg f(e) \end{aligned}$$

In the remaining equation, we know that e does not satisfy f, hence we have to check if there is another candidate further down in s:

$$\begin{aligned} \text{EXISTS}(f, \text{EMPTY}) &= \textbf{false} & \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \textbf{true} & \text{if } f(e) \\ \text{EXISTS}(f, \text{PUSH}(e, s)) &= \text{EXISTS}(f, s) & \text{if } \neg f(e) \end{aligned}$$

[1] Expression $\neg x$ is the negation of x.