Answers to quiz #2 on Algebraic Specification

Christian Rinderknecht

3 June 2005

1 Arrays

We want an algebraic specification of arrays. An array is a list whose size is fixed at the construction time and whose elements are items of the same type. There is a special item which is used as a default element when creating a new array (i.e. a freshly created array contains these default elements). Then the user can change an element to another value by specifying the integer index (i.e. position) in the array and the new value.
Let us call ITEM the specification of some items whose signature is as follows.

- **Defined types** The type of the items is noted t
- **Constructors**
  DEFAULT : t
  The term DEFAULT is a distinguished item, whose interpretation, i.e. meaning, is left to the user of this specification.

Let us call now ARRAY(ITEM) the specification of the arrays over items of type ITEM.t. The signature is as follows.

- **Defined types** The type of the arrays is noted t.
- **Constructors**
  - EMPTY : t
    The term EMPTY represents the array which contains no element.
  - CREATE : int $\times$ int $\rightarrow$ t
    The term CREATE(x, y) denotes an array whose elements are indexed from x to y, if $x < y$, or from y to x otherwise. The type int denotes the set of positive integers. Integers x and y are called *bounds*. If $x < y$ then x is the *lower bound* and y is the *upper bound*.

    For example, CREATE$(3, 5)$ is an array whose elements are indexed by integers 3, 4 and 5. It contains three elements. These elements are all equal to ITEM.DEFAULT. Also CREATE$(5, 3)$ is the same as $\textsc{Create}(3,5)$. Therefore, in both cases, the lower bound is 3 and the upper bound is 5.

  - $\textsc{Set} : \mathsf{t} \times \mathsf{int} \times \textsc{Item}.\mathsf{t} \rightarrow \mathsf{t}$
    The term $\textsc{Set}(a,n,e)$ represents an array equal to array a except that the element at index n is e. If n is out of bounds, i.e. if n is not between the bounds of a, the result of $\textsc{Set}$ is unspecified. Same if a is empty.

- **Functions**

  - $\textsc{Lower} : \mathsf{t} \rightarrow \mathsf{int}$
    The call $\textsc{Lower}(a)$ is the lower bound of array a, i.e. the smallest valid index. If a is empty, the result is unspecified.
  - $\textsc{Upper} : \mathsf{t} \rightarrow \mathsf{int}$
    The call $\textsc{Upper}(a)$ is the upper bound of array a, i.e. the greatest valid index. If a is empty, the result is unspecified.
  - $\textsc{Get} : \mathsf{t} \times \mathsf{int} \rightarrow \textsc{Item}.\mathsf{t}$
    The call $\textsc{Get}(a,n)$ denotes the element in array a at index n. If n is out of bounds or if a is empty, the result of $\textsc{Get}$ is unspecified.
  - $\textsc{Nth} : \mathsf{t} \times \mathsf{int} \rightarrow \textsc{Item}.\mathsf{t}$
    The call $\textsc{Nth}(a,n)$ denotes the nth element in array a. If a is empty or n is out of bounds, the call is unspecified.
  - $\textsc{Mem} : \mathsf{t} \times \textsc{Item}.\mathsf{t} \rightarrow \mathsf{bool}$
    The call $\textsc{Mem}(a,e)$ is $\textsc{true}$ if e is an element of array a and $\textsc{false}$ otherwise.
  - $\textsc{Inv} : \mathsf{t} \rightarrow \mathsf{t}$
    The call $\textsc{Inv}(a)$ denotes the array whose elements are in the reverse order as find in a. For example, let a be the array graphically represented as $\boxed{e_1}\boxed{e_2}\boxed{e_3}\boxed{e_4}$, then $\textsc{Inv}(a)$ is represented as $\boxed{e_4}\boxed{e_3}\boxed{e_2}\boxed{e_1}$.
  - $\textsc{Undo} : \mathsf{t} \times \mathsf{int} \rightarrow \mathsf{t}$
    The call $\textsc{Undo}(a,n)$ represents the array a without the last modification (using $\textsc{Set}$) at index n. If a is empty or has not been modified at index n, the call equals a.

Question. Complete this signature with equations defining the functions $\textsc{Nth}$, $\textsc{Mem}$, $\textsc{Inv}$ and $\textsc{Undo}$ (remember you can use $\textsc{Lower}$, $\textsc{Upper}$ or $\textsc{Get}$ if you need).

Answer.

- $\textsc{Nth}$ can be expressed in terms of $\textsc{Get}$ as follows:

$$\textsc{Nth}(a,n) = \textsc{Get}(a, \textsc{Lower}(a) + n - 1), \qquad \text{if } a \neq \textsc{Empty}.$$

- $\textsc{Mem} : \mathsf{t} \times \textsc{Item}.\mathsf{t} \rightarrow \mathsf{bool}$

  The first argument of $\textsc{Mem}$ is an array, so we can start by envisaging the three constructors of arrays:

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \mathbf{?} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} \qquad \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

  The first equation refers to an empty array into which we look for some element, so this element, as well as any element, is not present, for sure, so:

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} & \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

  As far as the second equation is concerned, we know that an unmodified array contains, by construction, only $\textsc{Item}.\textsc{Default}$ elements. So we must consider two cases: $e = \textsc{Default}$ or not:

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} & \text{if } e = \textsc{Default} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} & \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} & \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

  Or, simply

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \mathbf{?} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} & \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} & \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

  Now, the first case is simple:

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \textsc{True} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \mathbf{?} & \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} & \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

  The second one also:

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \textsc{True} \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \textsc{False} & \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \mathbf{?} & \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

The last equation refers to the case of an array modified at index n, in which we search for element e_2. The first idea that should occur is to check whether the new element at index n, namely e_1, is e_2 or not. Indeed, if they are equal, the result should be TRUE. So

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} && \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \textsc{True} && \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
&&& \text{and } e_1 = e_2 \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
&&& \text{and } e_1 \neq e_2
\end{aligned}$$

Or, simply

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} && \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \textsc{True} && \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e), e) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
&&& \text{if } e_1 \neq e_2
\end{aligned}$$

Then

$$\begin{aligned}
\textsc{Mem}(\textsc{Empty}, e) &= \textsc{False} && \\
\textsc{Mem}(\textsc{Create}(x, y), \textsc{Default}) &= \textsc{True} && \\
\textsc{Mem}(\textsc{Create}(x, y), e) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\textsc{Mem}(\textsc{Set}(a, n, e), e) &= \textsc{True} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Mem}(\textsc{Set}(a, n, e_1), e_2) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
&&& \text{and } e_1 \neq e_2
\end{aligned}$$

The last case refers to a modified array whose new element is different from which the one we look for. What happen if we simply ignore (i.e. undo) the last modification and search again? This means the right-hand side of the last equation would be $\textsc{Mem}(a, e_2)$.

This would be a problem if a contains e_2 at the same index n, i.e. contains a subterm like $\textsc{Set}(b, n, e_2)$. This is possible because our arrays remember all the modifications done since their creation. So, searching a instead of $\textsc{Set}(a, n, e_1)$ means searching in the previous modifications of the array $\textsc{Set}(a, n, e_1)$. But if e_2 was present at index

n in a, the value of $\textsc{Mem}(a, e_2)$ will be $\textsc{True}$, *despite we know that in the given array, there is no* e_2 *at index* n.

So we must find a way to ignore index n in the subsequent searches. The simplest way is to use an auxiliary function $\overline{\textsc{Mem}} : \mathsf{t} \times \textsc{Item}.\mathsf{t} \times \textsc{Stack}(\mathsf{int}).\mathsf{t} \rightarrow \mathsf{bool}$. The difference with $\textsc{Mem}$ is that $\overline{\textsc{Mem}}$ takes a supplementary argument which is a stack of indexes. These indexes are the already visited indexes, so they must be ignored in previous versions of the array, in order to avoid the problem we mentioned in the previous paragraph.

Therefore the definition of $\textsc{Mem}$ is now done in terms of $\overline{\textsc{Mem}}$:

$$\textsc{Mem}(a, e) = \overline{\textsc{Mem}}(a, e, \textsc{Stack}(\mathsf{int}).\textsc{Empty})$$

In order to save some space, let us write

- $n \in s$ instead of $\textsc{Stack}(\mathsf{int}).\textsc{Mem}(s, n)$ (we do not define this membership function on stack here, but is is easy to do);
- $n :: s$ instead of $\textsc{Stack}(\mathsf{int}).\textsc{Push}(n, s)$

Now we can rewrite our equations with $\overline{\textsc{Mem}}$ instead of $\textsc{Mem}$:

$$\begin{aligned}
\overline{\textsc{Mem}}(\textsc{Empty}, e, \mathsf{s}) &= \textsc{False} \\
\overline{\textsc{Mem}}(\textsc{Create}(x, y), \textsc{Default}, \mathsf{s}) &= \textsc{True} \\
\overline{\textsc{Mem}}(\textsc{Create}(x, y), e, \mathsf{s}) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\overline{\textsc{Mem}}(\textsc{Set}(a, n, e), e, \mathsf{s}) &= \textsc{True} && \textbf{if } \mathbf{n} \notin \mathbf{s} \\
& && \text{and } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\overline{\textsc{Mem}}(\textsc{Set}(a, n, e_1), e_2, \mathsf{s}) &= \textbf{?} && \text{if } e_1 \neq e_2 \\
& && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

We notice that we add a condition $n \notin s$ which means that the index n is visited here for the first time, so we can take the corresponding contents, e, into account (here the result is $\textsc{True}$).

What if $n \in s$? Actually, this situation encompasses whether $e_1 = e_2$ or not: in both cases, we must ignore the current index n and recursively explore the other indexes. This is achieved by the recursive call $\overline{\textsc{Mem}}(a, e_2, s)$. Technically, we just removed the $\textsc{Set}$ call of the lefthand side $\overline{\textsc{Mem}}(\underline{\textsc{Set}(}a,\underline{n,e_1)},e_2,s)$. We must add another equation:

$$\begin{aligned}
\overline{\textsc{Mem}}(\textsc{Empty},e,s) &= \textsc{False} \\
\overline{\textsc{Mem}}(\textsc{Create}(x,y),\textsc{Default},s) &= \textsc{True} \\
\overline{\textsc{Mem}}(\textsc{Create}(x,y),e,s) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e),e,s) &= \textsc{True} && \textbf{if n} \notin \textbf{s} \\
& && \text{and } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e_1),e_2,s) &= \overline{\textsc{Mem}}(a,e_2,s) && \textbf{if n} \in \textbf{s} \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e_1),e_2,s) &= \textbf{?} && \text{if } e_1 \neq e_2 \textbf{ and n} \notin \textbf{s} \\
& && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

Notice that the condition if $\textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)$ has been removed because it is a recursive call, so we just can let the non-recursive calls check the bounds (see the fourth equation). Note also that we had the condition $n \notin s$ to the last equation, because we just added the case $n \in s$ in the fourth equation.

The remaining case is also a recursive call because we know that $e_1 \neq e_2$, i.e. the element at index n is not the one we look for, and it is the first time we visit this index n, so we must record it in the stack s of indexes, so we will not consider it again (in the subsequent recursive calls):

$$\begin{aligned}
\overline{\textsc{Mem}}(\textsc{Empty},e,s) &= \textsc{False} \\
\overline{\textsc{Mem}}(\textsc{Create}(x,y),\textsc{Default},s) &= \textsc{True} \\
\overline{\textsc{Mem}}(\textsc{Create}(x,y),e,s) &= \textsc{False} && \text{if } e \neq \textsc{Default} \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e),e,s) &= \textsc{True} && \text{if } n \notin s \\
& && \text{and } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e_1),e_2,s) &= \overline{\textsc{Mem}}(a,e_2,s) && \text{if } n \in s \\
\overline{\textsc{Mem}}(\textsc{Set}(a,n,e_1),e_2,s) &= \overline{\textsc{Mem}}(a,e_2,n :: s) && \text{if } e_1 \neq e_2 \\
& && \text{and } n \notin s
\end{aligned}$$

- $\textsc{Inv} : \mathsf{t} \rightarrow \mathsf{t}$
  The sole argument of $\textsc{Inv}$ is an array, so we can simply start by enumerating all the array constructors:

$$\begin{aligned}
\textsc{Inv}(\textsc{Empty}) &= \textbf{?} \\
\textsc{Inv}(\textsc{Create}(x,y)) &= \textbf{?} \\
\textsc{Inv}(\textsc{Set}(a,n,e)) &= \textbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)
\end{aligned}$$

The first equation corresponds to the inversion of an empty array: it is pretty clear that it must be the empty array too:

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Create}(x,y)) &= \mathbf{?}\\ \textsc{Inv}(\textsc{Set}(a,n,e)) &= \mathbf{?} \qquad\qquad \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)\end{aligned}$$

The second equation refers to the case of an unmodified array to be inverted. Since, by construction, all elements of an unmodified array are ITEM.DEFAULT, the inverted array is the same as the original one, just as for the empty array. So

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Create}(x,y)) &= \textsc{Create}(x,y)\\ \textsc{Inv}(\textsc{Set}(a,n,e)) &= \mathbf{?} \qquad\qquad \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)\end{aligned}$$

The last equation refers to the case of an array modified at index n. The inverted array has thus to be modified symmetrically, i.e. it must have the same modification but on the symmetric index (starting from the end of the array instead of the beginning). The symmetric index of n in array a is

$$\textsc{Upper}(a) - n + \textsc{Lower}(a)$$

(Note that modifiying an element does not change the bounds.) So we have to do the modification

$$\textsc{Set}(\mathbf{?}, \textsc{Upper}(a) - n + \textsc{Lower}(a), e)$$

But what array do we must modify? It cannot be a because we have to invert the other elements in a before applying the last symmetric modification. So it must be $\textsc{Inv}(a)$. As a conclustion, we get

$$\begin{aligned}\textsc{Inv}(\textsc{Empty}) &= \textsc{Empty}\\ \textsc{Inv}(\textsc{Create}(x,y)) &= \textsc{Create}(x,y)\\ \textsc{Inv}(\textsc{Set}(a,n,e)) &= \textsc{Set}(\textsc{Inv}(a), \textsc{Upper}(a) - n + \textsc{Lower}(a), e)\\ &\qquad\qquad \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)\end{aligned}$$

Actually, we could simplify a little bit these equations by noting that if the array is not modified, then the inverted array is invariant. Formally:

$$\begin{aligned}\textsc{Inv}(a) &= a \qquad\qquad \text{if } a \neq \textsc{Set}(b,n,e)\\ \textsc{Inv}(\textsc{Set}(a,n,e)) &= \textsc{Set}(\textsc{Inv}(a), \textsc{Upper}(a) - n + \textsc{Lower}(a), e)\\ &\qquad\qquad \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a)\end{aligned}$$

- $\textsc{Undo} : \mathsf{t} \times \mathsf{int} \to \mathsf{t}$
  Since the first argument of $\textsc{Undo}$ is an array and since there are three array constructors, we start by enumerating these cases:

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \mathbf{?} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \mathbf{?} \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} \qquad \text{if } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

  The text says that if the array is empty or unmodified, the result of undoing the last modification is simply the original array:

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \textsc{Empty} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \textsc{Create}(x, y) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} && \text{if } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

  The remaining equation refers to the case of an array whose last modification has been made at index p. Since we want to undo the last modification at index n, we must check whether $p = n$ or not:

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \textsc{Empty} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \textsc{Create}(x, y) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} && \text{if } p = n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} && \text{if } p \neq n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

  Or, simply

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \textsc{Empty} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \textsc{Create}(x, y) \\
\textsc{Undo}(\textsc{Set}(a, n, e), n) &= \mathbf{?} && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} && \text{if } p \neq n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

  If the last modification was done at the index we want to undo, the result is the array without the call to $\textsc{Set}$:

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \textsc{Empty} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \textsc{Create}(x, y) \\
\textsc{Undo}(\textsc{Set}(a, n, e), n) &= a && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \mathbf{?} && \text{if } p \neq n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

Now, the last equation refers to the case where the last modification was done on a different index than the one we want to undo. Therefore, we must keep this modification and undo on the array *without* this modification:

$$\begin{aligned}
\textsc{Undo}(\textsc{Empty}, n) &= \textsc{Empty} \\
\textsc{Undo}(\textsc{Create}(x, y), n) &= \textsc{Create}(x, y) \\
\textsc{Undo}(\textsc{Set}(a, n, e), n) &= a && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \textsc{Set}(\textsc{Undo}(a, n), p, e) && \text{if } p \neq n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$

As a final note, we can simplify a little bit more the equations by grouping the two first ones:

$$\begin{aligned}
\textsc{Undo}(a, n) &= a && \text{if } a \neq \textsc{Set}(b, p, e) \\
\textsc{Undo}(\textsc{Set}(a, n, e), n) &= a && \text{if } \textsc{Lower}(a) \leqslant n \leqslant \textsc{Upper}(a) \\
\textsc{Undo}(\textsc{Set}(a, p, e), n) &= \textsc{Set}(\textsc{Undo}(a, n), p, e) && \text{if } p \neq n \\
&&& \text{and } \textsc{Lower}(a) \leqslant p \leqslant \textsc{Upper}(a)
\end{aligned}$$