In-situ Inversion of a Cyclic Permutation

by

W.H.J. Feijen, A.J.M. van Gasteren,
and D. Gries

Abstract

An algorithm is developed for the in-situ inversion of a cyclic permutation represented in an array. The emphasis is on the quo modo rather than on the quod; we are interested in finding concepts and notations for dealing more effectively with the formal development and proof of such algorithms, rather than in this particular algorithm itself.

Introduction

Let P be a permutation of the elements of a finite, nonempty universe, i.e. a one-to-one function from the universe onto the universe. The inverse permutation Q of P is defined by

$$Q.j = i \quad \equiv \quad P.i = j$$

for each i and j in the universe.

(Throughout, "." is used for function application.) We want an algorithm invert that changes an array H containing a permutation to its inverse:

(0) $\{H = P\}$ invert $\{H = Q\}$.

An algorithm for this problem is given in [1], but without explanation. This is typical of the current state of affairs with algorithms that deal with arrays in a complicated fashion. They are not explained at all, they are explained informally in terms of pictures, or they are explained more formally but in such a way that irrelevent, overwhelming detail crops up at the wrong places, making the proof less than convincing. (An example of the latter phenomenon appears in [0].)

In this note, we attempt to present some concepts and notation to make the development and proof of one such algorithm more convincing and appealing. We develop a solution to (0) similar to that of [1] but restricted to the case that P is a cyclic permutation, since this is where the heart of the problem lies.

Notation and Nomenclature

Consider a finite, nonempty universe. Elements of the universe are denoted by lower case letters, sequences of elements by capitals, the empty sequence by empty, and catenation of sequences (and elements) by juxtaposition.

For sequences, function rev is defined by

$$
\begin{aligned}
&\text{rev. empty} = \text{empty}\\
&\text{rev. } r = r\\
&\text{rev. } (X\,Y) = (\text{rev. } Y)\ (\text{rev. } X)\ .
\end{aligned}
$$

For any nonempty sequence X of distinct elements,

> $[X]$ is called a ring ; its elements are the elements of X .

By postulate, rings satisfy the

Rule of Rotation: $[X\ Y] = [Y\ X]$.

Relation between cyclic permutations and rings

A ring $[X]$ and a cyclic permutation P can be related using the convention that each element i of $[X]$ is followed by $P.i$. Of importance is the fact that the inverse Q of P is then likewise related to the ring $[rev.X]$:

$Q.j = i$
$=$ { definition of inverse }
$P.i = j$
$=$ { by the convention }
in $[X]$, i is followed by j
$=$ { by the definition of rev }
in $[rev.X]$, j is followed by i .

Having thus identified cyclic permutations and rings, we carry out the rest of the discussion in terms of rings .

Representational convention

We couple a ring h and an array H

using the following

Representation invariant: for each element r
and all sequences X and Y satisfying
$h = [X r Y]$,
$H.r =$ the first element of sequence YXr

Note that, with this convention, an application of the rule of rotation does not affect the value of array H.

Development of the algorithm

The specification

For ring h and array H coupled by the representation invariant, we wish to construct a program invert with functional specification

$$\{h = [U]\} \text{ invert } \{h = [\text{rev}.U]\}$$

and whose ultimate text is expressed in terms of H. Since rings are nonempty, we can write this as

$$\{h = [U\,p]\} \text{ invert } \{h = [p \text{ rev}.U]\}.$$

Further, for the moment, let us assume that h contains at least two elements, and let us develop program invert to satisfy

(1) $\{h = [U\,p\,q]\} \text{ invert } \{h = [q\ p \text{ rev}.U]\}$.

The loop invariant

To begin with, we choose as an intermediate state of invert a generalization of its initial and final state. We do so by introducing two sequences

X and Y and requiring that

$$P0: \quad h = [q\ X\ p\ Y]$$

be maintained. The initial state of invert (see (1)) then corresponds to $X = U$ and $Y = \text{empty}$: this follows from

$$
\begin{aligned}
& [q\ X\ p\ Y] \\
= & \quad \{ X = U \ \wedge\ Y = \text{empty} \} \\
& [q\ U\ p] \\
= & \quad \{ \text{rule of rotation} \} \\
& [U\ p\ q] .
\end{aligned}
$$

The final state of invert (see (1)) corresponds to $X = \text{empty}$ and $Y = \text{rev}.U$ (by substitution). The initial state can be transformed into the final state by shrinking X one element at a time until $X = \text{empty}$. To enforce that the final state satisfy $Y = \text{rev}.U$, we notice that initially $\text{rev}.X = \text{rev}.U$ and $Y = \text{empty}$ and require that

$$P1: \quad (\text{rev}.X)\ Y = \text{rev}.U$$

be maintained as well.

The algorithm

Using $P0$ and $P1$ as invariants and attempting to shrink X one element at a time leads to the algorithm

(2) $\quad X, Y := U, \text{empty}$
$\quad\quad \{\text{invariant}: \ P0 \wedge P1\}$
$\quad ;$ <u>do</u> $X \neq \text{empty}$
$\quad\quad \rightarrow$ <u>with</u> r and Z chosen to satisfy $X = r\,Z$:

massage h
; $X, Y := Z, rY$
od.

The invariance of $P1$ follows from the fact that for $X = rZ$,

$$
\begin{array}{ll}
 & wp(\text{"}X, Y := Z, rY\text{"}, P1) \\
= & \quad \{\text{axiom of assignment}\} \\
 & (\text{rev}.\, Z)\, rY = \text{rev}.\, U \\
= & \quad \{\text{definition of rev}\} \\
 & (\text{rev}.\,(rZ))\; Y = \text{rev}.\, U \\
= & \quad \{X = rZ\} \\
 & (\text{rev}.\, X)\; Y = \text{rev}.\, U \\
= & \quad \{\text{definition } P1\} \\
 & P1 \; .
\end{array}
$$

We still have to define massage h so that $P0$ is maintained by each loop iteration. From $P0$ and $X = rZ$ we conclude that its precondition is

$$h = [q\; r\; Z\; p\; Y],$$

and its postcondition is $wp(\text{"}X, Y := Z, rY\text{"}, P0)$, which is

$$h = [q\; Z\; p\; r\; Y] .$$

Hence, massage h has to satisfy

(3) $\{h = [q\; r\; Z\; p\; Y]\}$ massage h $\{h = [q\; Z\; p\; r\; Y]\}$.

Replacing thought variables by references to H

Our purpose now is to replace all references to variables h, U, X, Y, and Z of algorithm (2) by references to variables p, q, r, and H. (This is known as coordinate transformation.)

We first see how to implement massage h in terms of H. The representation invariant together with the precondition of (3) implies

$H.p$ = the first element of sequence Yq
$H.q = r$
$H.r$ = the first element of sequence Zp.

The representation invariant together with the postcondition of (3) implies

$H.p = r$
$H.q$ = the first element of sequence Zp
$H.r$ = the first element of sequence Yq.

This, together with the observation that the successors of the elements of Y and Z do not change, allows us to implement massage h in terms of H by (recall, that p, q, and r are distinct)

$$H.p, H.q, H.r := H.q, H.r, H.p .$$

Finally we observe, using the representation invariant, that

- in the initial state of invert (see (1)), $q = H.p$;
- by P0, the guard $X \neq$ empty is given by $H.q \neq p$; and
- by P0 and $X = rZ$, $r = H.q$.

Hence, thought variables h, U, X, Y, and Z can be eliminated, yielding the ultimate program:

```
{p is any element of the ring to be inverted}
q := H.p
; do H.q ≠ p
  → r := H.q ; H.p, H.q, H.r := H.q, H.r, H.p
  od .
```

Finally, it is simple to verify that the program is correct for a ring containing a single element.

Concluding remarks

The development of the algorithm consisted of introducing the ring as a suitable characterization of a cyclic permutation, coupling the ring and array representations using a representation invariant, developing an algorithm in terms of rings, and applying a coordinate transformation to arrive at an algorithm in terms of the array representation.

The non-standard activity in the development was the introduction of the notion ring and we will give -in short- the history of its invention. In a first effort to give a neat presentation of the above algorithm, the first two authors characterized the elements of a cyclic permutation in terms of its array representation H, using expressions like $H^k.p$. These expressions then diffused in vast numbers through the text, the mathematical formulae became almost unmanageable, and the resulting treatment miserably failed to convince. In a next effort, by the last two authors, the elements of a cyclic permutation were characterized in terms of a sequence s yielding expressions like s_i. Also in this case these expressions diffused in vast numbers through the text, and the resulting treatment -suffering from 'indexitis'- failed to convince almost equally miserably. The source of the trouble then became clear: the so-called natural, i.e. conventional, representations of cyclic permutations were not geared to our manipulative needs. The remedy then became clear as well: we had to abandon convention. We chose a different name for cyclic permutations – calling them rings – and started to design a ring calculus. The design of that calculus immediately revealed that for the sake of manageability few elements of rings should have a name. In the conventional notations of rings each element was named. In the presence of such overspecific nomenclature even a simple rule as the rule of rotation becomes awkward to formulate.

The above exercise again confirms many a computing scientist's impression that in designing algorithms the development of adequate mathematical notations is a key issue, an issue which is hardly addressed by traditional mathematics.

Acknowledgements

The members of the Eindhoven Tuesday Afternoon Club are acknowledged for some pertinent comments.

References

[0] Gries, D. The multiple assignment statement. IEEE Trans. Software Eng. SE-4, 2 (March 1978), 89-93.

[1] Huang, B.-C. An algorithm for inverting a permutation. IPL 12 (Oct 1981), 237-238

WF/AvG/DG, 16 August 1985