IN-SITU INVERSION OF A CYCLIC PERMUTATION[1]

W.H.J. Feijen[2]
A.J.M. van Gasteren[2]
D. Gries[3]

TR 85-703
September 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

[1] This research was supported by British Petroleum under its BP Venture Research Program and by the NSF under grant DCR-8320274.

[2] Dept. of Mathematics and Computing Science, University of Technology, 5600 MB EINDHOVEN, the Netherlands.

[3] Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA

In-situ Inversion of a Cyclic Permutation[1]

by

W.H.J. Feijen[2], A.J.M. van Gasteren[2], and D. Gries[3]

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 the *quod*; we are interested in finding concepts and notations for dealing more effectively with formal developments and proofs 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 $\overline{P}$ of P is the permutation defined by

$$\overline{P}.j = i \quad \equiv \quad P.i = j \qquad \text{for each } i \text{ and } j \text{ in the universe.}$$

(Throughout, "." is used for function application; $f.i$ is the result of applying function f to argument i.) We want an algorithm *invert* that changes an array H containing a permutation to its inverse[4]:

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

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 irrelevant, overwhelming detail crops up at the wrong places, making the proof less than convincing. (An example of the latter phenomenon appears in [0].)

[1] This research was supported by British Petroleum under its BP Venture Research Program and by the NSF under grant DCR-8320274.

[2] Dept. of Mathematics and Computing Science, University of Technology, 5600 MB EINDHOVEN, the Netherlands.

[3] Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA.

[4] Conventionally, an array H is thought of as a set of variables $H(i)$, where i ranges over some contiguous set of the integers. We prefer to think of an array as a function from its subscript values i to the array values $H.i$, and there is no reason, in general, to restrict its domain.

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, non-empty universe. Elements of the universe are denoted by lower case letters, sequences of elements by upper case letters, the empty sequence by *empty*, and catenation of sequences (and elements) by juxtaposition.

For sequences, function *rev* is defined by

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

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 $\overline{P}$ of P is then likewise related to the ring $[rev.X]$:

$\quad\overline{P}.j = i$
$=$ {definition of inverse}
$\quad P.i = j$
$=$ {by the convention}
$\quad$ in ring $[X]$, i is followed by j
$=$ {by the definition of *rev*}
$\quad$ in ring $[rev.X]$, 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 so that they represent the same cyclic permutation 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 $Y\ X\ r$.

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 specification

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

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

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

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

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

The loop invariant. To start, 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

$$P0\text{:}\ \ h = [q\ \ X\ \ p\ \ Y]$$

be maintained. The initial state of *invert* (see (1)) then corresponds to $X = U$ and $Y = empty$; this follows from

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

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

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

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

(2)
$$\begin{array}{l} \quad X,\ Y := U,\ empty \\ \quad \{invariant\text{:}\ P0\ \wedge\ P1\} \\ ;\ \mathbf{do}\ X \neq empty \\ \quad \rightarrow \mathbf{with}\ r \text{ and } Z \text{ chosen to satisfy } X = r\ Z\text{:} \\ \qquad\qquad massage\ h \\ \qquad\qquad ;X,\ Y := Z,\ r\ Y \\ \quad \mathbf{od}\ . \end{array}$$

The invariance of $P1$ follows from the fact that, for $X = r\ Z$,

$$
\begin{aligned}
& wp(\text{“}X,\ Y := Z,\ r\ Y\text{”},\ P1) \\
= & \ \{\text{axiom of assignment}\} \\
& (rev.Z)\ r\ Y = rev.U \\
= & \ \{\text{definition of } rev\} \\
& (rev.(r\ Z))\ Y = rev.U \\
= & \ \{X = r\ Z\} \\
& (rev.X)\ Y = rev.U \\
= & \ \{\text{definition of } P1\} \\
& P1\ .
\end{aligned}
$$

We still have to define *massage* h so that $P0$ is maintained by each loop iteration. From $P0$ and $X = r\ Z$ we conclude that its precondition is $h = [q\ r\ Z\ p\ Y]$, and its postcondition is $wp(\text{“}X,\ Y := Z, r\ Y\text{”}, P0)$, 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 a *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 $Y\ q$
$H.q = r$
$H.r$ = the first element of sequence $Z\ p$.

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

$H.p = r$
$H.q$ = the first element of sequence $Z\ p$
$H.r$ = the first element of sequence $Y\ q$.

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 represention 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 $X = r\ Z$, $r = H.q$.

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 a simple matter 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 representation 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 of a ring, and we give a short history of this introduction. 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 diffused in vast numbers throughout the text, the mathematical formulae became almost unmanageable, and the resulting treatment, suffering from "indexitis", failed miserably 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. These expressions also diffused in vast numbers throughout the text, and again indexitis led to a treatment that failed to convince.

The source of the trouble became clear: the conventional representations of cyclic permutations were not geared to our manipulative needs. The remedy then became clear as well: 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 for rings, *each* element is named, and, 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 that is hardly addressed by traditional mathematics.

Acknowledgements.

The members of the Eindhoven Tuesday Afternoon Club and the Cornell programming methodology group are acknowledged for 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.