The closure of a finite mapping

Geraint Jones
Programming Research Group
11 Keble Road
Oxford OX1 3QD

15 December 1988

This note is about my struggling to understand a clever algorithm for calculating what I suspect Richard Bird would call a homomorphic image of the range of the closure of a finite mapping.

Let f be a finite partial mapping $f : \mathcal{Z} \nrightarrow \mathcal{Z}$, and let *root* be a specific element of $\mathcal{Z}$. The image of *root* under the closure of f is the set $\{i : 0 \leq i : f^i(\mathit{root})\}$. A homomorphic image of a set s has the form $\bigoplus x : x \in s : g(x)$ for some associative, idempotent, commutative operator $\oplus$. The algorithm in question must calculate $\bigoplus i : 0 \leq i : g(f^i(\mathit{root}))$. To do this, it must visit each $f^i(\mathit{root})$ at least once, and must eventually be able to establish that it has done so.

Define $\overline{f}$, the closure of f, by

$$\overline{f}(x) \mathrel{\widehat{=}} \{i : 0 \leq i : f^i(x)\}$$

(I am trying to be consistent about notation: that right hand side is the set of all the $f^i(x)$ for which there is an i satisfying $0 \leq i$.) To be precise, of course, this definition should be something like

$$\overline{f}(x) \mathrel{\widehat{=}} \{i : 0 \leq i \wedge (\forall j : 0 \leq j < i : f^j(x) \in \operatorname{dom} f) : f^i(x)\}$$

but I hope you trust me to be a little careless, for economy of paper. If f is finite, which is to say that if $\operatorname{dom} f$ is finite, each $\overline{f}(x)$ is also finite. Define

$$\begin{array}{lcl} \mathit{finite}(f, x) & \widehat{=} & \exists i : 0 \leq i : f^i(x) \notin \operatorname{dom} f \\ \mathit{loops}(f, x) & \widehat{=} & \exists i, j : 0 \leq j < i : f^i(x) = f^j(x) \end{array}$$

then since f is finite it follows that either $\mathit{finite}(f, \mathit{root})$ or $\mathit{loops}(f, \mathit{root})$.

The arguments about the algorithms can be simplified by considering the total function $\hat{f}$ defined by

$$\hat{f}(x) \mathrel{\widehat{=}} \begin{cases} f(x) & x \in \operatorname{dom} f \\ x & x \notin \operatorname{dom} f \end{cases}$$

because $\overline{\hat{f}} = \overline{f}$ but $\mathit{loops}(\hat{f}, \mathit{root})$.

I will describe three algorithms for calculating $\overline{f}(\mathit{root})$, two poor ones that I understand how to develop, and one good one which I can only present by pulling it like a rabbit out of a hat.

The naïve algorithm

It takes little effort to derive the algorithm that constructs the whole of the image of *root* under the closure of f. The obvious thing (it seems to me) to do is to consider the $\hat{f}^i(root)$ in turn, because

$$\begin{array}{rcl} loops(\hat{f}, root) & \iff & \exists i, j : 0 \leq j < i : \hat{f}^i(root) = \hat{f}^j(root) \\ & \iff & \exists i : 0 \leq i : \exists j : 0 \leq j < i : \hat{f}^i(root) = \hat{f}^j(root) \\ & \iff & \exists i : 0 \leq i : \hat{f}^i(root) \in \{j : 0 \leq j < i : \hat{f}^j(root)\} \end{array}$$

The 'standard rabbit' to use here is: introduce a variables for a constant; specifically introduce s to be the set $\{j : 0 \leq j < i : \hat{f}^j(root)\}$ and x to be the $\hat{f}^i(root)$ in question. (That is probably really introducing i for the constant infinity in the ranges of the existential quantification.) An invariant for the loop in the obvious program will include

$$\begin{array}{rcl} \exists i : 0 \leq i : a & = & h(s) \\ \wedge\ s & = & \{j : 0 \leq j < i : \hat{f}^j(root)\} \\ \wedge\ x & = & \hat{f}^i(root) \end{array}$$

where $h(s) \mathrel{\widehat{=}} \bigoplus y : y \in s : g(y)$.

An adequate variant for this program will be $\#(\overline{f}(root) - s)$, whence

$a, s, x := g(root), \{root\}, \hat{f}(root)$
while $x \notin s$ **do** $a, s, x := a \oplus g(x), s \cup \{x\}, \hat{f}(x)$

establishes $a = h(\overline{f}(root))$.

The problem with this program is that it carries around an essentially unbounded set of integers. For example, if this were a mechanism for visiting a collection of files in a file system it would almost certainly have to record the set s by storing a bit with each file in the filesystem. Fortunately, there are obvious ways of getting the same result with only a finite amount of state.

A slow algorithm

The choice of variant for the last program could be relaxed. Since $\#\,\overline{f}(root)$ is bounded by $\#\,\mathrm{dom}\,f + 1$, you could take $\#\,\mathrm{dom}\,f - \#\,s$, although there is no reason to suppose that s ever gets anything like as large as $\mathrm{dom}\,f$.

Thinking along these lines, you can decide to introduce a variable to represent not $\{j : 0 \leq j < i : \hat{f}^j(root)\}$ but rather $\{j : 0 \leq j < i : j\}$. Of course, you can represent this set just by the value of i. The invariant for this program will include $x = \hat{f}^i(root)$, and $a = h(\{j : 0 \leq j < i : \hat{f}^j(root)\})$.

In the course of the proof you need to observe that

$$\#\,\{x : x \in s : f(x)\} < \#\,s \ \Rightarrow\ \exists x, y : x, y \in s : x \neq y \wedge f(x) = f(y)$$

That is another 'standard rabbit', called the pigeonhole principle or something like that. From this you can argue

$$\begin{aligned}
i > \#\,\mathrm{dom}\, f \quad &\Rightarrow \quad \#\{j : 0 \le j \le i : \hat{f}^j(\mathit{root})\} \le \#\,\mathrm{dom}\, f + 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad < \#\{j : 0 \le j \le i : j\} \\
&\Rightarrow \quad \#\{j : 0 \le j \le i : \hat{f}^j(\mathit{root})\} < \#\{j : 0 \le j \le i : j\} \\
&\Rightarrow \quad \exists j : 0 \le j < i : \hat{f}^i(\mathit{root}) = \hat{f}^j(\mathit{root}) \\
&\Rightarrow \quad \overline{f}(\mathit{root}) \subseteq \{j : 0 \le j < i : \hat{f}^j(\mathit{root})\}
\end{aligned}$$

An adequate variant is very easy, say $\#\,\mathrm{dom}\, f - i$, whence

$a, i, x := g(\mathit{root}), 1, \hat{f}(\mathit{root})$
while $i \le \#\,\mathrm{dom}\, f$ **do** $a, i, x := a \oplus g(x), i + 1, \hat{f}(x)$

establishes $a = h(\overline{f}(\mathit{root}))$.

The problem this time is that it can take an awful long time for the loop to terminate. In the worst case, it can take $\#\,\mathrm{dom}\, f$ iterations, however small a part of $\mathcal{Z}$ is accessible from *root*. You can try to improve this by substituting a variable for the $\mathrm{dom}\, f$, for example

$a, i, x, \mathit{min}, \mathit{max} := g(\mathit{root}), 1, \hat{f}(\mathit{root}), \mathit{root}, \mathit{root}$
if $x < \mathit{min}$ **then** $\mathit{min} := x$
if $x > \mathit{max}$ **then** $\mathit{max} := x$
while $i \le (\mathit{max} - \mathit{min})$ **do begin**
    $a, i, x := a \oplus g(x), i + 1, \hat{f}(x)$
    if $x < \mathit{min}$ **then** $\mathit{min} := x$
    if $x > \mathit{max}$ **then** $\mathit{max} := x$
end

maintains

$$\{j : 0 \le j < i : \hat{f}^j(\mathit{root})\} \subseteq \overline{f}(\mathit{root}) \subseteq \{x : \mathit{min} \le x \le \mathit{max} : x\}$$

I was rather proud of the fact that the expected behaviour of that algorithm is better, if there is some locality in the values of f. It is still possible, however, for this to take a time proportional to $\#\,\mathrm{dom}\, f$.

A clever algorithm

The trouble is that there is an algorithm that takes a time proportional to $\#\,\overline{f}(\mathit{root})$, the size of the part of the mapping accessible from *root*, but that I do not know how to 'discover' it. For that reason, this section comes backwards: first the algorithm, then the invariant, then the proof.

Let $n > 1$, then the algorithm is

$$
\begin{array}{l}
a, i, \mathit{back}, \mathit{front} := g(\mathit{root}), 1, \hat{f}(\mathit{root}), \hat{f}(\mathit{root}) \\
\textbf{while } i < n \textbf{ do } a, i, \mathit{front} := a \oplus g(\mathit{front}), i+1, \hat{f}(\mathit{front}) \\
\textbf{while } \mathit{front} \neq \mathit{back} \textbf{ do begin} \\
\qquad i, \mathit{back} := 0, \hat{f}(\mathit{back}) \\
\qquad \textbf{while } i < n \textbf{ do } a, i, \mathit{front} := a \oplus g(\mathit{front}), i+1, \hat{f}(\mathit{front}) \\
\textbf{end}
\end{array}
$$

The invariant for the principal loop certainly includes

$$
\begin{array}{rrcl}
\exists i : 0 < i : & \mathit{front} & = & \hat{f}^{ni}(\mathit{root}) \\
\wedge & \mathit{back} & = & \hat{f}^{i}(\mathit{root}) \\
\wedge & a & = & h(\{j : 0 \leq j < ni : \hat{f}^{j}(\mathit{root})\})
\end{array}
$$

and the trick is in showing that

$$
\begin{array}{rcl}
\mathit{loops}(\hat{f}, \mathit{root}) & \Rightarrow & \exists i, j : 0 \leq j < i : \hat{f}^{i}(\mathit{root}) = \hat{f}^{j}(\mathit{root}) \\
& \Rightarrow & \exists i, j : 0 \leq j \wedge 0 < i : \hat{f}^{i}(\hat{f}^{j}(\mathit{root})) = \hat{f}^{j}(\mathit{root}) \\
& \Rightarrow & \exists i, j : 0 \leq j \wedge 0 < i : \\
& & \quad \forall n, k : 0 \leq n \wedge j \leq k : \hat{f}^{ni}(\hat{f}^{k}(\mathit{root})) = \hat{f}^{k}(\mathit{root})
\end{array}
$$

Take, for example, the minimum $k \geq j$ for which i divides k, and conclude that

$$
\begin{array}{l}
\mathit{loops}(\hat{f}, \mathit{root}) \\
\qquad \Rightarrow \quad \exists k : 0 < k : \forall n : 0 \leq n : \hat{f}^{nk}(\hat{f}^{k}(\mathit{root})) = \hat{f}^{k}(\mathit{root}) \\
\qquad \Rightarrow \quad \forall n : 1 < n : \exists k : 0 < k : \hat{f}^{nk}(\mathit{root}) = \hat{f}^{k}(\mathit{root})
\end{array}
$$

The converse proof is easier, for $n > 1 \wedge i > 0 \Rightarrow ni > i \geq 0$ so

$$
\hat{f}^{ni}(\mathit{root}) = \hat{f}^{i}(\mathit{root}) \quad \Rightarrow \quad \overline{f}(\mathit{root}) \subseteq \{j : 0 \leq j < ni : \hat{f}^{j}(\mathit{root})\}
$$

The real beauty of this algorithm is that its running time is proportional to the size of the part of the mapping f that is accessible from *root*. Define

$$
\begin{array}{rcl}
\mathit{leadin} & \hat{=} & \min i : 0 \leq i : \exists j : i < j : \hat{f}^{i}(\mathit{root}) = \hat{f}^{j}(\mathit{root}) \\
\mathit{period} & \hat{=} & \min i : 0 < i : \exists j : 0 \leq j : \hat{f}^{i+j}(\mathit{root}) = \hat{f}^{j}(\mathit{root})
\end{array}
$$

then you can show that the number of iterations of the outer loop is bounded by

$$
\min j : \mathit{leadin} \leq j : \mathit{period} \mid (n-1)j
$$

which is certainly bounded by $\mathit{leadin} + \mathit{period}$, which is $\#\,\overline{f}(\mathit{root})$, so the number of iterations of the inner loop is certainly bounded by $n \times \#\,\overline{f}(\mathit{root})$.

Envoi

The clever algorithm is surprisingly sensitive to the initialisation. My first attempt to reconstruct it from memory went wrong because I wrote a program something like

$$
\begin{array}{l}
a, i, \mathit{back}, \mathit{front} := g(\mathit{root}), 1, \mathit{root}, \hat{f}(\mathit{root}) \\
\textbf{while } \mathit{front} \neq \mathit{back} \textbf{ do begin} \\
\quad i, \mathit{back} := 0, \hat{f}(\mathit{back}) \\
\quad \textbf{while } i < n \textbf{ do } a, i, \mathit{front} := a \oplus g(\mathit{front}), i+1, \hat{f}(\mathit{front}) \\
\textbf{end}
\end{array}
$$

which maintained the invariance of

$$
\begin{array}{rrcl}
\exists i : 0 \leq i : & \mathit{front} & = & \hat{f}^{ni+1}(\mathit{root}) \\
\wedge & \mathit{back} & = & \hat{f}^{i}(\mathit{root}) \\
\wedge & a & = & h(\{j : 0 \leq j < ni : \hat{f}^{j}(\mathit{root})\})
\end{array}
$$

and this one only works for general f and *root* only in case $n = 2$. (Of course, for efficiency reasons, you would probably not want to implement this algorithm for $n > 2$.) The proof of termination involves finding an element of

$$\{j : \mathit{lead} \leq j : (n-1)j = 1 \pmod{\mathit{period}}\}$$

which might be empty, for example if both $n - 1$ and *period* were even. I confess myself mildly surprised when that went wrong: that will teach me to wave hands in the common room.

Each of the algorithms can be made to terminate sooner when $\mathit{finite}(f, \mathit{root})$, provided the program can test for membership of the domain of f. I could not face the details of the proofs of those programs.

Finally, I think the clever algorithm is relatively well known, but I could not think where to look for it.

The problem

Ah, well, the real reason for giving you this is that I am dissatisfied with the conjuring trick of producing the algorithm more or less fully formed from a recess of my memory. The party line seems to be that algorithms can be developed by pulling out of the hat only well-known rabbits. My problem is that I do not know the rabbit needed to arrive at the clever algorithm.