Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 225 (2009) 83–98
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Definite Descriptions and Dijkstra's Odd Powers of Odd Integers Problem

Hugh Gibbons[1]

Computer Science Dept.,
Trinity College,
Dublin, Ireland

Abstract

The use of Frege-Russell style definite descriptions for giving meaning to functions has been long established and we investigate their use in the development of Functional Programs and from these to the development of correct imperative programs. In particular, we investigate the development of a functional program for a problem, "Odd powers of odd integers", discussed by Dijsktra. If the correctness of termination is not a concern then it is straightforward to develop a partially correct program. Further properties of the specification are needed to develop a totally correct program.

Keywords: definite descriptions, functional programming, assertions, partial and total correctness.

1 Introduction

The use of definite descriptions dates back to Frege and Russell [13] and also to further development by Quine [12] and Scott [14]. The use and definitions of definite descriptions are explained in Kalish and Montague [9]. In this article we consider reusing definite descriptions in the development of functional programs. As assertions have a central role in the development of imperative programs as promoted by Dijkstra [5] and Gries [8] and the Refinement Calculus [11], we consider the role of definite descriptions in the development of functional programs which can then be further developed to imperative programs.

In this article we investigate in detail the formal development of totally correct programs for a specification described by Dijkstra [6].

For $1 \leq p$ and odd p and $1 \leq k$ and odd r such that $1 \leq r < 2^k$, a value x exists such that

$$1 \leq x < 2^k \wedge 2^k | (x^p - r) \wedge odd\ x$$

[1] Email: hugh.gibbons@cs.tcd.ie

doi:10.1016/j.entcs.2008.12.068

We are using '$|$' for 'divides'.

If proof of termination is not a concern then a very straightforward partially correct program can be easily developed. Proving termination uses induction and from the induction proof a simple functional program is derived. Based on this functional program, a totally correct imperative program is developed similar to that given by Dijkstra. Further properties of the specification are derived using the functional program and an alternative functional program is derived. While the alternative functional program is straighforward, its development into an imperative program is not. This development involves the tranformation of linear recursion to an appropriate tail recursive form which can then be directly transformed to imperative programs with the tail recursive programs providing the invariants for the associated loops.

In the development of the functional versions of the specification we make use of the definite descriptors, '*the*' and '*least*'.

1.1 Definite description, 'the'

$$
\begin{aligned}
y &= the\,one\,and\,only\,item\,satisfying\,p\\
y &= (the\,x \bullet p\,x)\\
&\quad \{Russell\}\\
&\equiv (\forall x \bullet x = y \equiv p\,x)\\
&\quad \{Quine\}\\
\{y\} &= \{x \bullet p\,x\}
\end{aligned}
$$

1.2 Definite description, 'least'

$$
\begin{aligned}
y &= (least\,x \bullet n \le x \wedge p\,x)\\
&\equiv n \le y \wedge p\,y \wedge (\forall x \bullet n \le x < y \to \neg(p\,x))
\end{aligned}
$$

In particular,

$$
\begin{aligned}
g\,n &= (least\,x \bullet n \le x \wedge p\,x)\\
&= (p\,n \to n) \wedge (\neg(p\,n) \to (least\,x \bullet (n+1) \le x \wedge p\,x)\\
&= if\ p\,n\ then\ n\ else\ g\,(n+1)
\end{aligned}
$$

In terms of functional programming lists,

$$
\begin{aligned}
y &= (least\,x \bullet n \le x \wedge p\,x)\\
y &= head[x \mid x \leftarrow [n..],\, p\,x]
\end{aligned}
$$

1.3 Simple Floor Square Root

As an introductory application of definite descriptions we develop a simple program for finding the integer square root of a number.

For $0 \le x$ we define the (positive) square root of x, as

$$\sqrt{x} = (the\,r \bullet 0 \le r \wedge r^2 = x)$$

We can define the floor square root, $\lfloor\sqrt{x}\rfloor$ as

$$\begin{aligned} r = \lfloor\sqrt{x}\rfloor \\ &\equiv r \leq \sqrt{x} < r+1 \\ &\equiv r^2 \leq x < (r+1)^2 \end{aligned}$$

$$\text{i.e. } \lfloor\sqrt{x}\rfloor = (the\, r \bullet 0 \leq r \wedge r^2 \leq x < (r+1)^2)$$

Show that

$$\lfloor\sqrt{x}\rfloor = (least\, r \bullet 0 \leq r \wedge x < (r+1)^2)$$

Theorem 1.1

$$y = (least\, r \bullet 0 \leq r \wedge x < (r+1)^2) \Rightarrow y \in \{r \bullet 0 \leq r \wedge r^2 \leq x < (r+1)^2\}$$

Proof Since

$$\begin{aligned} y = (least\, r \bullet 0 \leq r \wedge x < (r+1)^2) \\ &\Rightarrow 0 \leq y \wedge x < (y+1)^2 \end{aligned}$$

Need to just show that $y^2 \leq x$

$$\begin{aligned} y = (least\, r \bullet 0 \leq r \wedge x < (r+1)^2) \\ &\Rightarrow (\forall r \bullet 0 \leq r < y \rightarrow \neg(0 \leq r \wedge x < (r+1)^2) \\ &\equiv (\forall r \bullet 0 \leq r < y \rightarrow (0 > r \vee x \geq (r+1)^2)) \\ &\equiv (\forall r \bullet 0 \leq r < y \rightarrow x \geq (r+1)^2) \\ &\quad \{witness\ \hat{r} = y-1\} \\ &\Rightarrow x \geq y^2 \end{aligned}$$

End proof. □

Let

$$\begin{aligned} fl\_sqrt\, x\, n &= (least\, r \bullet n \leq r \wedge x < (r+1)^2) \\ &= if\ x < (n+1)^2\ then\ n\ else\ fl\_sqrt\, x\, (n+1) \end{aligned}$$

$$therefore$$

$$fl\_sqrt\ x\ 0 = \lfloor\sqrt{x}\rfloor$$

Writing this as a functional program:

$$\begin{aligned} &\lfloor \sqrt{x} \rfloor \ \equiv\ fl\_sqrt\ x\ 0 \\ &\quad \textbf{where} \\ &\qquad fl\_sqrt\ x\ n \\ &\qquad \mid\ x\ <\ (n+1)^2\ \equiv\ n \\ &\qquad \mid\ x\ \geq\ (n+1)^2\ \equiv\ fl\_sqrt\ x\ (n+1) \end{aligned}$$

We can rewrite this tail recursive functional program as an imperative program with the loop invariant based directly on the functional program.

$\lfloor\sqrt{-}\rfloor$`::Real` $\rightarrow$ `Int`

```
⌊√x⌋ ≡
    { Pre: 0 ≤ x }
     local
      n:Int
     begin
      n := 0
      {Inv: fl_sqrt x 0 = fl_sqrt x n}
      while x ≥ (n+1)^2 do
       n := n+1
      end {fl_sqrt x 0 = n}
      {⌊√x⌋ = n}
      Result := n
     end.
```

Here the tail recursive program, *fl_ sqrt*, is used as a loop invariant. This connection between tail recursion and loop invariants is further developed in Gibbons [7] and in Broy and Krieg-Bruckner [3].

2 Dijkstra's Odd Powers of Odd Integers

For clarity, we repeat the Dijkstra specification given above.
For $1 \le p$ and odd p and $1 \le k$ and odd r such that $1 \le r < 2^k$, a value x exists such that

$$1 \le x < 2^k \wedge 2^k|(x^p - r) \wedge odd\ x$$

Example 2.1 13 is a witness for the existential quantifier x in $(\exists x \bullet 1 \le x < 2^4 \wedge 2^4|(x^3 - 5) \wedge odd\, x)$ as

$$\begin{aligned} 2^4|(13^3 - 5) &\equiv 16|(2197 - 5) \\ &\equiv 2192 = 137 \times 16 \end{aligned}$$

A witness for x in $(\exists x \bullet 0 \le x \wedge 2^k|(x^p - r))$ must be odd as if $2^k\,|\,(x^p - r)$ then $x^p - r$ is even, hence x^p is odd since r is odd, therefore x is odd.

2.1 Definite Description function

Consider the following function, *f*, described by a definite description for finding a witness for x,

Let $pre\ p\ r\ k\ = 1 \le p \wedge odd\ p \wedge 1 \le k \wedge odd\ r \wedge 1 \le r < 2^k$

$$pre\ p\ r\ k\ \rightarrow$$
$$f\ p\ r\ k\ \equiv\ (least\ x \bullet 1 \le x < 2^k \wedge odd\ x \wedge 2^k|\,(x^p - r))$$

i.e. using a precondition

$$\{\ pre\ p\ r\ k\ \}$$
$$f\ p\ r\ k\ \equiv\ (least\ x \bullet 1 \le x < 2^k \wedge odd\ x \wedge 2^k|\,(x^p - r))$$

We rewrite $f\ p\ r\ k$ as a functional program. using list comprehension,

$$\{\ pre\ p\ r\ k\ \}$$
$$f\ p\ r\ k\ \equiv\ head[x \mid x\ \leftarrow\ [1..2^k],\ odd\ x,\ 2^k|\ (x^p - r)]$$

or using the higher order function, *filter*

$$\{pre\ p\ r\ k\ \}$$
$$f\ p\ r\ k\ \equiv\ head\ (filter\ b\ [1, 3, ..])$$
where
$$b\ x\ \equiv\ x\ <\ 2^k\ \wedge\ 2^k|\ (x^p - r)$$

Since in general,

$$g\ n \equiv (least\ x\ \bullet\ n \leq x\ \wedge\ b\ x)$$
$$\equiv if\ b\,n\ then\ n\ else\ g\,(n+1)$$

we get an alternate (tail recursive) functional program, f_1,

$$\{\ pre\ p\ r\ k\ \}$$
$$f_1\ p\ r\ k\ \equiv\ f_{loc}\ 1$$
where
$$f_{loc}\ x$$
$$|\ 2^k|(x^p - r)\ \wedge\ x\ <\ 2^k \equiv x$$
$$|\ otherwise \qquad \equiv f_{loc}\ (x+2)$$

We can use Quickcheck [4] to test if the functions f and f_1 are the same.

We can rewrite the functional program f_1 as an iterative imperative program: see figure Algorithm 1 (Imperative f1)

Algorithm 1 Imperative f1

```
f1::Int×Int×Int → Int
f1 (p,r,k) ≡
{Pre: 1 ≤ p ∧ 1 ≤ k ∧ 1 ≤ r < 2^k ∧ odd r }
    local
     x:Int
    begin
     x := 1
     {Inv: f_loc 1 = f_loc x }
     while ¬( 2^k|(x^p − r) ∧ x < 2^k ) do
      x := x+2
     end {f_loc 1 = x }
     {f_1 p r k = x}
     { 2^k|(x^p − r) ∧ x < 2^k ∧ odd x }
     Result := x
    end.
```

The loop in this imperative program iterates through the odd integers until it reaches an x such that $2^k|(x^p - r) \wedge x < 2^k$. When tested the program halts for the given inputs but testing is not enough to prove correctness. If the loop terminates then the program will give the correct result.

3 Alternative FP Version

To show that the loop terminates we are back to showing

$$(\exists x \bullet odd\, x \wedge 1 \leq x < 2^k \wedge 2|(x^p - r))$$

under the assumption

$$Pre\, p\, r\, k: \; 1 \leq p \wedge odd\, p \wedge 1 \leq k \wedge odd\, r \;\wedge\; 1 \leq r < 2^k.$$

A normal strategy in the context of the refinment calculus is to strenghten the precondition but here the precondition is weakened by dropping the conjunct,
$1 \leq r < 2^k$ as it can be shown that $(\exists x \bullet 1 \leq x \wedge 2^k|(x^p - r))$ from the weaker assumption

$$Pre'\, r\, k: \; 1 \leq p \wedge odd\, p \wedge 1 \leq k \;\wedge\; odd\, r$$

Whatever satisfies Pre also satisfies Pre', i.e. $Pre \Rightarrow Pre'$.
Later we will show that the least witness, $\hat{x}$, for $(\exists x \bullet 1 \leq x \wedge 2^k|(x^p - r))$ is such that $\hat{x} < 2^k$.

Theorem 3.1

$$1 \leq p \wedge odd\, p \wedge 1 \leq k \;\wedge\; odd\, r \Rightarrow (\exists x \bullet odd\, x \;\wedge\; 2^k|\,(x^p - r))$$

Proof (By induction on k)

Base case (k=1)

Let $x = 1$; since r is odd then $1^p - r$ is even therefore $2|\,(1^p - r)$

Also, x = 1 is the least such x.

Induction step:

Assume x is the least x such that $odd\, x \;\wedge\; 2^k|\,(x^p - r)$, determine least $odd\, y$ such that $2^{k+1}|(y^p - r)$.

If $2^{k+1}|\,(x^p - r)$, let $y = x$

If $\neg(2^{k+1}|\,(x^p - r))$then $\frac{x^p - r}{2^k}$ is odd.

Let $y = x + 2^k$,

$$\begin{aligned}
&\tfrac{y^p - r}{2^k} \\
=\;& \tfrac{(x+2^k)^p - r}{2^k} \\
=\;& \tfrac{x^p + p\, x^{p-1} 2^k + ... + p\, x\, 2^{(p-1)k} + 2^{pk} - r}{2^k} \\
=\;& \tfrac{x^p - r}{2^k} + p\, x^{p-1} + ... + p\, x\, 2^{(p-2)k} + 2^{(p-1)k}
\end{aligned}$$

$$\{ \frac{x^p - r}{2^k} \; and\; p\, x^{p-1} \; are\; odd \; \}$$

$$\frac{y^p - r}{2^k} \; is\; even$$

$$2^{k+1} | \, (y^p - r)$$

Also, $y = x + 2^k$ is the least such y as if $y = x + n$ with (even n) and $n < 2^k$then

$$\begin{array}{ll} (x+n)^p - r & \\ = & \frac{x^p - r}{2^k} + \frac{p\, x^{p-1} n + \ldots + n^p}{2^k} \\ = & \frac{x^p - r}{2^k} + \frac{n\,(p\, x^{p-1} + \ldots + n^{p-1})}{2^k} \end{array}$$

$2^k | \, (x^p - r)$ but $\neg(2^k | n\,(px^{p-1} + \ldots + n^{p-1}))$ as $n < 2^k$ and $p\, x^{p-1} + \ldots + n^{p-1}$ is odd. □

From this inductive proof, we get the recursive functional program, $f_2\, p\, r\, k$, for finding x such that $odd\, x \;\wedge\; 2^k | \, (x^p - r)$

$$\begin{array}{l} \{ \;\leq\; p \;\wedge\; odd\; p \;\wedge\; 1 \;\leq\; k \;\wedge\; odd\; r \,\} \\ f_2\; p\; r\; 1 \;\equiv\; 1 \\ f_2\; p\; r\; (k+1) \;\equiv\; if\; 2^{k+1} | (x^p \;-\; r)\; then\; x\; else\; x + 2^k \\ \quad \textbf{where} \\ \qquad x \;\equiv\; f_2\; p\; r\; k \end{array}$$

It is clear this function terminates with respect to the precondition: $1 \leq k$.

Theorem 3.2

$$\begin{array}{l} pre\; p\, r\, k \quad \Rightarrow \quad f_2\, p\, r\, k < 2^k \\ \qquad\qquad where \\ \qquad\qquad\qquad pre\; p\, r\, k \equiv\; 1 \leq p \wedge odd\, p \wedge 1 \leq k \wedge odd\, r \;\wedge\; 1 \leq r < 2^k \end{array}$$

Proof (By induction on k)

Base case: (k=1)

$$\begin{aligned} f_2\, p\, r\, 1 &= 1 \\ &< 2 \end{aligned}$$

Induction Step: $(k > 1)$ Assume $f_2\, p\, r\, k < 2^k$

Case $2^{k+1} | \, f_2\, p\, r\, k$

$$\begin{aligned} f_2\, p\, r\, (k+1) &= f_2\, p\, r\, k \\ &< 2^k \\ &< 2^{k+1} \end{aligned}$$

Case $\neg(2^{k+1} | \, f_2\, p\, r\, k)$

$$\begin{aligned} f_2\, p\, r\, (k+1) &= f_2\, p\, r\, k + 2^k \\ &< 2^k + 2^k \\ &= 2^{k+1} \end{aligned}$$

End Proof. □

The function, $f_2\ p\ r\ k$, also satisfies the stronger specification :

$$1 \leq p \wedge odd\, p \wedge 1 \leq k \wedge odd\, r \ \wedge\ 1 \leq r < 2^k \rightarrow$$
$$f_2\, p\, r\, k \equiv (least\, x \bullet 1 \leq x < 2^k \ \wedge\ odd\, x \ \wedge\ x^p\, mod\, 2^k = r)$$

and thus is a functional program that satisfies the specification given by Dijkstra.

Dijkstra provides an imperative solution based on the invariant

$$1 \leq x < 2^k \wedge 2^k | (x^p - r) \wedge odd\, x$$

Concerning his own imperative solution, Dijkstra states in [6]:

> "I have evidence that, despite the existence of this very simple solution, the problem is not trivial: many computer scientists could not solve the programming problem within an hour. Try it on you colleagues, if you don't believe me"

We derive an iterative solution from the recursive version $f_2\ p\ r\ k$.

3.1 Iterative version

The imperative program, f_1 above, may be considered an imperative solution of the original f once termination has been guaranteed.

A more direct version of an iterative program can be developed from the recursive program f_2. Consider the set

$$F = \{((p, r, k), y) \bullet 1 \leq k \wedge odd\, p \wedge 1 \leq p \wedge odd\, r \wedge y = f_2\, p\, r\, k\}$$

Let $pre\ p\ r\ k\ =\ 1\ \leq\ k\ \wedge\ odd\ p\ \wedge\ 1\ \leq\ p\ \wedge\ odd\ r\ \wedge\ 1\ le\ r$
For $k\ =\ 1$

$$((p, r, 1), 1) \in F.$$

If $((p, r, k), x) \in F$ then

$$\begin{array}{l} \textbf{if } 2^{k+1} | (x^p - r) \textbf{ then} \\ \quad ((r, k+1), x) \in\ F \\ \textbf{else} \\ \quad ((p, r, k+1), x + 2^k) \in\ F \end{array}$$

The set F is an inductively defined set of ordered pairs such that

$$((p, r, k), y) \in F \Rightarrow y = f_2\, p\, r\, k$$

Based on the inductive set F we get the specification for an iterative function f_t

$$\begin{array}{l} 1\ \leq\ k\ \wedge\ odd\ p, r\ \wedge\ 1\ \leq\ p, r\ \rightarrow \\ f_t\ p\ n\ r\ k\ x\ \equiv\ (least\ y\ \bullet\ k = n\ \wedge ((p, r, k), y) \in\ F \wedge\ x = y) \end{array}$$

We write $f_t\ p\ n\ r\ k\ x$ as the functional program,

$$\begin{array}{ll} \{\ odd\ p\ \wedge 1\ \leq\ k \wedge\ odd\ r\ \} & \\ f_t\ p\ n\ r\ k\ x & --\{\ x\ =\ f_2\ p\ r\ k\ \wedge\ 1 \leq\ k \leq\ n\ \} \end{array}$$

$$
\begin{array}{lll}
\mid & k = n & \equiv x \\
\mid & 2^{k+1} | (x^p - r) & \equiv f_t\ n\ r\ (k+1)\ x \\
\mid & \textit{otherwise} & \equiv f_t\ p\ n\ r\ (k+1)\ (x + 2^k)
\end{array}
$$

Rewriting this as an imperative program; see Algorithm 2 (Imperative ft),

Algorithm 2 Imperative ft

```
ft::Int×Int×Int → Int
ft (p,r,k) ≡
{Pre: 1 ≤ p ∧ odd p ∧ 1 ≤ k ∧ odd r ∧ 1 ≤ r < 2^k }
local
 j, x : Int
begin
 x := 1; j := 1
 {Inv: 2^k|(x^p − r) ∧ odd x ∧ 1 ≤ j ≤ k}
 while j ≠ k do
  if 2^(j+1)|(x^p − r) then
   j := j+1
  else
   x := x + 2^j;
   j := j+1
  end
 end
 { 2^k|(x^p − r) ∧ odd x }
 Result := x
end.
```

Rather than explicitly using 2^k we can calculate it implicitly as in the following: see Algorithm 3 (Dijkstra version). This is the version similar to that developed by Dijkstra and like the version developed here does not make use of the restriction $1 \leq r < 2^k$ in the initial precondition.

Algorithm 3 Dijkstra version

```
ft::Int×Int×Int → Int
ft (p,r,k) ≡
{Pre:  1 ≤ p ∧ odd p ∧ 1 ≤ k ∧  odd r ∧ 1 ≤ r < 2^k}
    local
     j,x,d : Int
    begin
      j := 1; x := 1; d := 2
     {Inv: 2^j|(x^p − r) ∧ odd x ∧ 1 ≤ j ≤ n ∧ d = 2^j}
     while j ≠ k do
      if ¬(2 ∗ d|(x^p − r)) then
       x := x+d
      end
      d := 2*d
      j := j+1
     end
     { 2^k|(x^p − r) ∧ odd x }
     Result := x
    end.
```

4 Linear Recursion

As a consequence of the following theorem the restriction in the precondition of the specifiction that $1 \leq r < 2^k$ is redundant. A linear recursive function results which also satifies the Dijkstra specification and which then can be developed into an imperative program.

Theorem 4.1

$$2^k|(x^p - r) \equiv 2^k|(x^p - (r \; mod \; 2^k))$$

Proof For some q_1 and q_2,

$$
\begin{aligned}
& 2^k|(x^p - r) \\
& \qquad \equiv (x^p - r) = q_1\,2^k \\
& \qquad \equiv \{\, r = q_2\,2^k + r\,mod\,2^k \,\} \\
& \qquad \quad (x^p - (q_2\,2^k + r\,mod\,2^k)) = q_1\,2^k \\
& \qquad \equiv x^p - r\,mod\,2^k = q_1\,2^k + q_2\,2^k \\
& \qquad \equiv x^p - r\,mod\,2^k = (q_1 + q_2)2^k \\
& \qquad \equiv 2^k|(x^p - r\,mod\,2^k)
\end{aligned}
$$

End Proof. □

From this theorem can conclude that

$$f_2\,p\,r\,k = f_2\,p\,(r\,mod\,2^k)\,k$$

Since $r\, mod\, 2^k < 2^k$ we also have,

$$\{\, 1 \leq p \wedge odd\, p \wedge 1 \leq r \wedge odd\, r \,\}$$
$$f_2\, p\; r\; k < 2^k$$

therefore, the restriction that $r < 2^k$ is redundant.

If $x = f_2\; p\; r\; k$ then $x < 2^k \wedge x^p\; mod\; 2^k = r\; mod\; 2^k$

4.1 Alternative program

Taking advantage of the result that $f_2\; p\; r\; k = f_2\; p\; (r\; mod\; 2^k)\; k$ and without loss of generality fixing p to be the odd number 3 we can rewrite $f_2\; 3$ as a new function f_3 where

$$
\begin{array}{l}
\{\, 1 \leq k \wedge odd\; r \,\} \\
f_3\; r\; 1 \equiv 1 \\
f_3\; r\; (k+1) \equiv if\; 2^{k+1} | (x^3 - r)\; then\; x\; else\; x + 2^k \\
\quad \textbf{where} \\
\qquad r_1 \equiv mod\; r\; 2^k \\
\qquad x \equiv f_3\; r_1\; k
\end{array}
$$

This program, f_3 is more difficult to transform to an imperative/iterative version. In order to derive an imperative version we use the result that f_3 can be rewritten in a linear recursive form by progressive transformations.

Define auxillary functions

$$dv\; x\; r\; k \equiv if\; 2^k \,|\, (x^3 - r)\; then\; 0\; else\; 1$$

and

$$next\; x\; r\; k \equiv x + (dv\; x\; r\; k) * 2^{k-1} \quad -- \; finds\; the\; next\; terms\; after\; x$$

then we can rewrite f_3 as

$$
\begin{array}{l}
\{\, 1 \leq k \wedge odd\; r \,\} \\
f_3\; r\; 1 \equiv 1 \\
f_3\; r\; (k+1) \equiv next\; x\; r\; (k+1) \\
\quad \textbf{where} \\
\qquad r_1 \equiv mod\; r\; 2^k \\
\qquad x \equiv f_3\; r_1\; k
\end{array}
$$

Writing this in an 'if - then - else' format we get

$$
\begin{array}{ll}
\{\, 1 \leq k \wedge odd\; r \,\} & \\
f_3\; r\; k \equiv & if\;\; k \neq 1 \\
 & then\; next\; (f_3\; (mod\; r\; 2^{k-1})\; (k-1))\; r\; k \\
 & else\; 1
\end{array}
$$

Using ordered pairs and auxillary functions

$$nt\; (xr,\; xk)\; (yr, yk) \equiv (next\; xr\; yr\; yk,\; yk)$$

$$gt\ (r,\ k)\ \equiv\ (mod\ r\ 2^{k-1}\ ,\ k-1)$$
$$bt\ (r,k)\ \equiv\ k\ \neq\ 1$$

we can reduce this further to a standard form.

$$f_3\ x\ \equiv\ if\ bt\ x\ then\ nt\ (f_3\ (gt\ x))\ x\ else\ x$$

4.2 Transforming Linear Recursion

Termination of the linear recursive function, lr,

$$lr\ x\ \equiv\ if\ b\ x\ then\ n\ (lr\ (g\ x))\ x\ else\ x$$

depends on the existence of an number $i \geq 0$ such that $\neg\, b\,(g^i\ x)$ where $g^0\ x\ =\ x$ and $g^{i+1}\ x\ =\ g\ (g^i\ x)$.

For a binary function f, similar to definitions in Bird [2], we will use the following higher order function $\backslash f$ 'left-reduce'

$$\backslash f\ [\,] = [\,]$$
$$\backslash f\ [x] = x$$
$$\backslash f\ [x_1 \ldots x_n] = f\ (\backslash f\ [x_1 \ldots x_{n-1}])\ \ x_n$$

In particular,

$$\backslash f\ [x_1, x_2, x_3] = f\ (f\ x_1\, x_2)\, x_3$$

If we have an infix operator $\oslash$, not necessarily associative, then

$$\backslash \oslash [x_1 \ldots x_n] = (..(x_1 \oslash x_2) \ldots) \oslash x_n$$

Given the sequence or list
$gs\ =\ [g^i\ x,\ g^{i-1}\ x,\ \ldots,\ g\ x,\ x]\ where\ i\ =\ (least\ j\ \bullet\ \neg\, b\ (g^j\ x))$
then the linear recursive function,

$$lr\, x = if\ \ b\, x\ then\ (n\, (lr\, (g\, x))\, x)\ else\ x$$

can be implemented as

$$lr\, x = \backslash n\ gs$$

A more general version of this result is proved in Gibbons [7] which is related to the approach of the Computer aided Intuition guided Programming (CIP) group in Munich Technical University led by Bauer [1].

Theorem 4.2

$$\begin{aligned} lr\, x\ \ &=\ \ \backslash n\ gs \\ &\quad where \\ &\qquad i = (least\, j\ \bullet \neg b\, (g^j\, x)) \\ &\qquad gs\ =\ [g^i x,\ g^{i-1}x, \ldots g\, x,\ x] \end{aligned}$$

Proof (By Induction)

Notation:
If $i < 0$ then $[g^i x,\ g^{i-1}x, \ldots g\, x,\ x] = []$
If $i = 0$ then $[g^i x,\ g^{i-1}x, \ldots g\, x,\ x] = [x]$
end Notation

$i = 0$

$0 = (least\, j\ \bullet \neg b\,(g^j\, x))$
tf. $\neg b\,(x)$
tf. $lr\ x = x$
Also, $\backslash n\, gs = x$
tf. $lr\ x = \backslash n\, gs$

$i > 0$, Assume true for $i - 1$, show true for i.

$i = (least\, j\ \bullet \neg b\,(g^j\, x))$
tf. considering $g\, x$
$i - 1 = (least\, j\ \bullet \neg b\,(g^j\,(g\, x)))$
Let $gs1 = [g^{i-1}(g\, x),\ g^{i-2}(g\, x), \ldots g\, x]$
By induction,
$\backslash n\ gs1 = lr\,(g\, x)$
Since $i > 0$

$$
\begin{aligned}
lr\, x &= n\,(lr\,(g\, x))\, x \\
&= n\,(\backslash n\ gs1)\, x \\
&\quad \{defn.\ \backslash n\,\} \\
&= \backslash n\ gs
\end{aligned}
$$

End proof. □

4.2.1 Implementing $lr\, x$

Since $lr\, x = \backslash n\,[g^i x,\ g^{i-1}x \ldots g\, x,\ x]$ where $= (least\, j \bullet \neg b\,(g^j\, x))$ we consider implementing $\backslash n\,(x : xs)$.

For an item x and a list xs , define a function lrt via

$$lrt\, x\, xs = \backslash n\,(x : xs)$$

tf.

$$
\begin{aligned}
lrt\, x\, [] &= \backslash n\,[x] \\
&= x
\end{aligned}
$$

Also, for $xs = y : ys \neq []$,

$$
\begin{aligned}
lrt\, x\, xs &= \backslash n\,(x : (y : ys)) \\
&\quad \{prop.\ \backslash n\} \\
&= \backslash n((n\, x\, y) : ys) \\
&= lrt\,(n\, x, y)\, ys
\end{aligned}
$$

The function, lrt, is the tail recursive function

$lrt\ x\ xs\ =\ if\ xs\ \neq\ [\,]\ then\ lrt\ (n\ x\ (head\ xs))\ (tail\ xs)\ else\ x$

which can be rewritten as an imperative program which we can use to write an imperative program for lrt.(Algorithm 4)

Algorithm 4

```
lrti x ≡
{ Pre:  gs = [g^i x, g^(i-1) x, ... g x, x] }
 local
  y:Int;
  ys:[Int]
 begin
  y := head gs; ys := tail gs
  {Inv:  \n gs = \n(y : ys)}
  while  ys ≠ [] do
   y := n y (head ys)
   ys := tail ys
  end
  {y = lrt x }
  Result := y
  end lrti
```

4.2.2 Finalising Implementation

What is still needed is a program to establish

$$
\begin{aligned}
gs \quad &= \quad [g^i x,\, g^{i-1}x, \ldots g\, x,\, x] \\
&\quad where \\
&\qquad\quad i = (least\, j\ \bullet \neg b\,(g^j\, x))
\end{aligned}
$$

Notation:
For lists xs, ys
$xs\ +\!\!+\ ys$ is the concatenation of the lists.
end Notation

In a similar way to implementing $\backslash n\, gs$ we consider implementing the function

$$p\ x\ xs\ =\ [g^{i-1}x,\ \ldots\ g\ x,\ x]\ +\!\!+\ xs$$

as for each $0 \leq j < i$ we have $b\,(g^j\, x)$.
If $\neg b\, x$ then $i = 0$ and therefore $p\ x\ xs\ =\ xs$.
If $b\, x$ then

$$
\begin{aligned}
p\ (g\ x)\ (x : xs) &= [g^{i-2}(g\ x),\ \ldots\ g\ x]\ +\!\!+\ x : xs \\
&= [g^{i-1}x,\ \ldots\ g\ x,\ x]\ +\!\!+\ xs \\
&= p\ x\ xs
\end{aligned}
$$

We can write $p\; x\; xs$ as a tail recursive function

$p\; x\; xs \;\equiv\; \mathit{if}\; b\; x\; \mathit{then}\; p\;(g\; x)\;(x : xs)\; \mathit{else}\; xs$

Based on this function we can write the following imperative program, init_lrt, (Algorithm 5) that will establish

$$
\begin{aligned}
gs \quad &= \quad [g^i x,\; g^{i-1} x, \ldots g\, x,\; x] \\
&\quad \mathit{where} \\
&\qquad\qquad i = (\mathit{least}\, j \;\bullet \neg b\,(g^j\, x))
\end{aligned}
$$

Algorithm 5

```
init_lrt : Int -> [Int]
   init_lrt x ≡
    { Pre: (∃i • i = (least j • ¬b (g^j x)) }
    local
     y:Int;
     gs:[Int]
    begin
     y := x; gs := []
     while (b y) do    {Inv: p x [] = p y gs }
      gs := y:gs
      y := g y
     end {y = g^i x }
     gs := y:gs
     {Post:gs = [g^i x, g^(i-1) x, ... g x, x] }
     Result := gs
    end init_lrt
```

5 Conclusion

Based on Dijkstra's specification of the problem of 'Odd Powers of Odd Integers' this article applies the theory of definite descriptions and functional programming to first develop a correct functional program and from this to the development of a correct imperative program. If the correctness of termination is not a concern then it is straightforward to develop a partically correct imperative program. By developing functional programs many properties of the program are established and while Dijkstra develops a totally correct program via his own weakest precondition technique it is not clear how other properties could be established. Here it is shown that

$$f\, p\, r\, k = f\, p\,(r\; \mathit{mod}\; 2^k)\; k$$

and hence the restriction of $r < 2^k$ is redundant which is not noted by Dijkstra.

The development of the totally correct functional programs was done independently of Dijstra's article and it was the discovery of Dijkstra's article that motivated the more complete development presented here. Including the development of the functional programs clarifies the development of the imperative program and this article agrees with the view of Manna and Waldinger [10] who state that

"Recursion seem to be the ideal vehicle for systematic program construction".

In this article recursion is also used as the vehicle for the development of the loop invariants of imperative programs and as a result integrates the development of imperative programs with that of functional programs.

References

[1] Bauer, F. and H. Wossner, "Algorithmic Language and Program Development," Springer-Verlag, 1981.

[2] Bird, R., *An introduction to the theory of lists*, in: M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, Springer-Verlag, 1987 .

[3] Broy, M. and B. Krieg-Bruckner, *Derivation of invariant assertions during program development by transformation*, ACM Transactions on Programming Languages and Systems **2** (1980).

[4] Claessen, K. and J. Hughes, *Specification-based testing with quickcheck*, in: J. Gibbons and O. de Moor, editors, *the fun of programming*, Palgrave Macmillan, 2003 .

[5] Dijkstra, E., "A Discipline of Programming," Prentice Hall, 1976.

[6] Dijkstra, E., "Selected Writings on Computing: A Personal Perspective," Springer-Verlag, 1982.

[7] Gibbons, H., "Integrating Relational and Imperative Programming via a Weakest Precondition Calculus," Ph.D. thesis, Trinity College Dublin (1990).

[8] Gries, D., "The Science of Programming," Springer-Verlag, 1981.

[9] Kalish, D., R. Montague and G. Mar, "Logic Techniques and Formal Reasoning," Harcourt Brace Jovanovich, 1980.

[10] Manna, Z. and R. Waldinger, *Synthesis: Dreams => programs*, Technical report, Standford Research Institute (SRI) (1977).

[11] Morgan, C., "Programming from Specifications," Prentice-Hall, 1990.

[12] Quine, W., "Set Theory and its Logic," Harvard, 1969.

[13] Russell, B., *On denoting*, in: A. Martinich, editor, *The Philosophy of Language*, Oxford University Press, 1996 .

[14] Scott, D., *Existence and description in formal logic*, in: K. Lambert, editor, *Philosophical Applications of Free Logic*, Oxford University Press, 1991 .