Declarative View of Imperative Programs

Hugh Gibbons,

Trinity College,
Dublin,
Ireland.

Abstract

By giving a declarative meaning to an imperative program, the verification of the imperative program is switched from the imperative paradigm to the declarative or logic paradigm where one can take advantage of, for example, referential transparency. Rather than 'compiling' an imperative program to a 'lower level' we 'inverse compile' the imperative program to the 'higher' declarative level. The declarative view of an imperative program is a predicate associated with the imperative program such that if this predicate satisfies the specification of the program then the imperative program is correct relative to the specification. In one sense the associated predicate gives a declarative meaning to the imperative program.

"… and I do consider assignment statements and pointer variables to be among computer science's 'most valuable treasures'."

D. Knuth [5]

1. Introduction

In imperative programming, programs are implicitly state transformations. We want to abstract away from dealing with states and try to express programs within the "problem domain", that is, we want to give a declarative meaning to imperative programs. If we want to sort a list, our program for doing this should deal with lists and use the properties of lists. Logic/Functional programming and also programming with assertions allow us to do this abstraction. Imperative programs may have functional and relational features - functions and procedures - but they also contain constructs which deal only with state transformation e.g. the assignment statement.

"Not only goto statements are being questioned: we hear complaints about floating-point calculations, global variables, semaphores, pointer variables, and even assignment statements."

D. Knuth [6]

We are assuming a structured programming language without "goto" and the aim is to transform a structured program to a program without an assignment. We are not just complaining about the 'assignment statement', our aim is to show how it can be removed from programming. Without the assignment statement we gain the advantage of referential transparency ('substitution of equals for equals') which allows both for easier program verification and easier program construction. Also, by transforming a structured program to an 'assignment-less' program we indicate a non-operational declarative meaning to the program.

2. Associating a Predicate with an Imperative Program.

In the axiomatic semantics of Hoare/Dijkstra/Gries, a predicate pair gives the meaning of the imperative program and in this context a program is viewed as a predicate transformer, transforming a predicate to a predicate. For example, if we have a program, MOD, that calculates the 'integer modulo function' then the predicate pair

Pre(a,b): $a \geq 0 \wedge b > 0$, Post(z,a,b): $z = a \bmod b$

would give the meaning to the program, MOD.
In Hoare triple form we have

$\{Pre(a,b)\}$ MOD $\{Post(z,a,b)\}$.

In the notation of the Refinement Calculus [Morgan], we can write this as

$z:[Pre(a,b),\ Post(z,a,b)] \sqsubseteq$ MOD

To prove the program, MOD, correct we would show that

$\vdash \{Pre(a,b)\}$ MOD $\{Post(z,a,b)\}$

i.e. $\vdash Pre(a,b) \rightarrow wp(MOD, Post(z,a,b))$.

where wp(S,Q) is the weakest precondition (wp) of program S with respect to the predicate Q. (See Appendix 1.)

In determining wp(MOD, Post(z,a,b)) we are finding the wp of the program MOD relative to the particular postcondition, Post(z,a,b). We consider a more general use of wp that will give a declarative meaning to a program such as MOD. We consider the conditions where a program will halt. If the program halts then the output program variable, z, will have some value, which we can denote (using the convention of the Z notation) by z!.

If a program, PROG, halts i.e. $\vdash wp(PROG,true)$ then

$$
\begin{array}{lll}
wp(PROG,true) & \vdash & wp(PROG, (\exists z! \mid: z = z!)) \\
 & & \{\text{Deterministic programs}\} \\
 & = & (\exists z! \mid wp(PROG, z = z!))
\end{array}
$$

Our aim is to associate the predicate wp(PROG, z = z!) with the program PROG and show how, in particular, this gives an appropriate meaning to the program for proving the program correct. We first consider when the program, PROG, is just the assignment statement.

2.1 Assignment and Substitution

The Assignment statement x := e maps the state s to $s\dagger[x \mapsto s(e)]$, where † is the function over-ride operator. In particular, $s\dagger[x \mapsto s(e)](x) = s(e)$, i.e. the value of x in the new state is the value of e in the previous state. From this it can be shown, but not here, that

$wp(x := e, Q(x)) = Q(e)$

Using Lambda Calculus notation we can express

$$
\begin{array}{lll}
wp(x := e, Q(x)) & = & \text{Let } x := e \text{ in } Q(x) \\
 & = & (\lambda x)(Q(x))e
\end{array}
$$

In the Lambda Calculus when making a substitution, we have to be careful not to make a 'free' variable bound. With weakest precondition we have an analogous problem.

Consider

$wp(x := x+1, z = x)$.

In particular,

$\vdash z = x+1 \rightarrow wp(x := x+1, z = x)$

because

$wp(x := x+1, z = x) \equiv z = x+1$

Even though $wp(x := x+1, z = x)$ is a predicate we cannot directly use the

<u>Substitution Rule</u>: $(z)(z = e \rightarrow P(z)) = P(e)$.

From

$\vdash z = x+1 \rightarrow wp(x:=x+1, z = x)$

using the rule unrestrictedly, we could get

$\vdash wp(x:=x+1, x+1 = x)$ -- z is replaced by x+1

$\vdash wp(x:=x+1, False)$

$\vdash False$ -- a contradiction.

Using the Lambda Calculus formalism,
from

$\vdash z = x+1 \rightarrow (\lambda x)(z = x)(x+1)$

we cannot conclude

$\vdash (\lambda x)(x+1 = x)(x+1)$

as the predicate z = x is not 'free for x', as we are replacing z by x+1 in which x occurs free but after replacement the x becomes bound by (λx). But we can change the bound variable x to x' so that
from

$\vdash z = x+1 \rightarrow (\lambda x)(z = x)(x+1)$

we can conclude

$\vdash z = x+1 \rightarrow (\lambda x')(z = x')(x+1)$

$\vdash (\lambda x')(x+1 = x')(x+1)$

$\vdash x+1 = x+1.$

In an analogous way, using x', we get

$\vdash z = x+1 \rightarrow wp(x:=x+1, z = x)$

$\vdash z = x+1 \rightarrow wp(x':=x+1, z = x')$

$\vdash wp(x':=x+1, x+1 = x')$ -- subst. Rule

$\vdash x+1 = x+1.$

Substitution Capture Rule for wp:

If e in x:=e does not contain the variable x then
from

$\vdash z = e \rightarrow wp(x:=e, x = z)$

conclude

$\vdash wp(x:=e, x = e).$

More generally,

if S does not contain z and does not change any of the variables in e then

from

$\vdash z = e \rightarrow wp(S, y = z)$

conclude

$\vdash wp(S, y = e).$

2.2 wp as an Evaluating Mechanism

As a simple illustration, consider the simple program,

```
Swap:     x := a; y := b;
          t := x;
          x := y;
          y := t;
```

We can use wp to find the values of x and y after Swap has been executed. Let x! and y! be the (‘out’) values of x and y after executing Swap.

By calculating wp for assignment and compostion (see Appendix 1) we find that

$wp(Swap, x = x! \wedge y = y!) = x! = b \wedge y! = a$

tf. (therefore) $\vdash x! = b \wedge y! = a \rightarrow wp(Swap, x = x! \wedge y = y!)$

Since x!, y! do not occur in Swap and Swap does not change the values of a,b, we get by “Substitution Capture Rule for wp”

$\vdash wp(Swap, x=b \wedge y=a).$

As another example, using an conditional statement, consider

MIN: Init: x:=a; y:=b;
IF: **if** x < y **then** z:=x **else** z:=y **fi**

To find the value of z after executing MIN we use wp(MIN, z = z!).

$$\begin{aligned} wp(IF, z = z!) &= (x < y \wedge wp(z := x, z = z!)) \vee (x \geq y \wedge wp(z := y, z = z!)) \\ &= (x < y \wedge z! = x) \vee (x \geq y \wedge z! = y) \end{aligned}$$

With Initialisation, Init, we get

$$wp(MIN, z = z!) = (a < b \wedge z! = a) \vee (a \geq b \wedge z! = b).$$

In this example, it may be better to consider MIN declared as the procedure

```
MIN(in X,Y :integer; out Z :integer) is
     begin
           IF: if X < Y then Z := X else Z := Y fi
     end
```

with the call

MIN(a,b,z) equivalent to the program MIN above.

In this case we associate with the procedure call, MIN(a,b,z), the predicate

wp(MIN(a,b,z), z = z!) getting $(a < b \wedge z! = a) \vee (a \geq b \wedge z! = b)$

2.3 The Associated Predicate of the Imperative Program, MIN.

This approach gives us a way of associating a predicate with an imperative program.

Given the procedure MIN(**in** X, Y:integer, **out** Z:integer) we can associate with the procedure call, MIN(x,y,z), the predicate/logic program, Min(x,y,z!), where

Min(x,y,z!) = wp(IF, z=z!) -- IF is the procedure body of MIN.

and the query/call to the predicate program

$$\begin{aligned} Min(a,b,z!) &= wp(x := a; y := b, \ Min(x,y,z!)) \\ &= wp(x := a; y := b; MIN(x,y,z), \ z = z!) \\ &= wp(MIN(a,b,z), z = z!) \end{aligned}$$

Since the predicate/logic program, $Min(x,y,z!) = (x < y \wedge z! = x) \vee (x \geq y \wedge z! = y)$

we can write this in clause form as,

$Min(x,y,z!) \leftarrow x<y \wedge z! = x.$

$Min(x,y,z!) \leftarrow x\geq y \wedge z! = y.$

i.e.

$Min(x,y,x) \leftarrow x<y.$
$Min(x,y,y) \leftarrow x\geq y.$

More generally, let

in $\underline{X}$ be the **in** (read-only) parameters,

in out $\underline{Y}$ be the **in out** (read-write) parameters

and **out** $\underline{Z}$ be the **out** (write) parameters

then with the procedure declaration

P(**in** $\underline{X}$:Tx; **in out** $\underline{Y}$:Ty; **out** $\underline{Z}$:Tz)

we can associate the predicate,

$p(\underline{x},\underline{y},\underline{z}!) = wp(\ P(\underline{x},\underline{y},\underline{z}),\ \underline{z} = \underline{z}!)$

with the procedure call,

P(x,y,z).

i.e. the predicate, $p(\underline{x},\underline{y},\underline{z}!)$, which is defined as $wp(\ P(\underline{x},\underline{y},\underline{z}),\ \underline{z} = \underline{z}!)$ is associated with the procedure call, $P(\underline{x},\underline{y},\underline{z})$.

We are using the following notation conventions,

we use capitals for imperative programs and lower case for predicates,
we are using $\underline{x}$ to stand for a sequence of variables.

Later, in discussing particular examples, we relax these conventions for convenience.

2.3.1 Extended example, $z = m^n$

Consider the procedure declaration

```
      POWER(in  M:REAL; in N:NATURAL; out Z:REAL) is
            local
                  X: REAL;
                  K :NATURAL
            begin
                        { M ε ℝ ∧ N ε ℕ }
              INIT:     X := M;  K := N;
                        Z := 1;
              LOOP:     while K ≠ 0 loop
                           if ODD(K) then K := K-1; Z := Z*X fi;
                           K := K/2;  X := X*X
                        end loop
                        { Z = M^N }
            end
```

We associate the predicate p(m,n,z!) = wp(POWER(m,n,z), z = z!) with the procedure call POWER(m,n,z).
We consider this in more detail, in particular, we consider how to associate a predicate with a loop statement.

2.3.2 Associating a Predicate with a Loop

To associate a predicate with a loop statement consider the abstract form

F: **while** B **loop** S

Since F is a function on states we can write this as

F = **if** B **then** S;F **else** SKIP **fi**.

Associate with F the predicate

$$\begin{aligned} f(\underline{x},\underline{z!}) &= wp(F, \underline{z} = \underline{z!}) \\ &= (B \wedge wp(S, wp(F, \underline{z} = \underline{z!}))) \vee (\neg B \wedge \underline{z} = \underline{z!}) \\ &= (B \wedge wp(S, f(\underline{x}, \underline{z!}))) \vee (\neg B \wedge \underline{z} = \underline{z!}). \end{aligned}$$

where $\underline{x}$ are the **in** variable and $\underline{z}$ are the **out** variables

As an example let us consider the relation associated with the above program, Power, for finding m^n. Find wp(POWER(m,n,z), z=z!).

wp(POWER(m,n,z), z = z!) = wp(INIT;LOOP, z = z!)

Let

$$\begin{aligned} L(x, k, z, z!) &= wp(LOOP, z=z!) \\ &= k \neq 0 \wedge wp(BODY, wp(LOOP, z = z!)) \\ &\quad \vee\ k = 0 \wedge z = z! \\ &= k \neq 0 \wedge wp(BODY,\ L(x, k, z, z!)) \\ &\quad \vee k = 0 \wedge z = z! \\ &= k \neq 0 \wedge (odd(k) \wedge L(x*x, (k-1)/2, z*x, z!) \\ &\quad \vee\ \neg odd(k) \wedge L(x*x, k/2, z, z!)) \\ &\quad \vee k = 0 \wedge z = z! \end{aligned}$$

tf.

$$\begin{aligned} L(x, k, z, z!) = \ & k = 0 \wedge z = z! \\ & \vee\ k \neq 0 \wedge \neg\ odd(k) \wedge L(x*x, k/2, z, z!) \\ & \vee k \neq 0 \wedge odd(k) \wedge L(x*x, (k-1)/2, z*x, z!) \end{aligned}$$

Let $p(m, n, z!) = wp(POWER(m, n, z), z = z!)$
$= wp(M:=m; N:=n; INIT, L(x, k, z, z!))$
$= L(m, n,1, z!)$

We associate with the procedure call, POWER(m, n, z) the predicate p(m,n,z!).

2.4 Program Verification

We can use the associated relation of an imperative program to prove it correct. For example, consider using this approach to verify a program MOD for calculating (a mod b) a,b ε Z (integers) which we define as

$$r = a \bmod b \quad \text{iff} \quad (\exists k:\mathbb{N} \mid: a = k*b + r) \wedge 0 \le r<b$$

From this definition of mod, (a mod b) is defined only when b > 0 and when a ≥ 0, therefore we take $a \geq 0 \wedge b>0$ as the precondition.

```
MOD(in A,B:integer; out Z:integer) is
     { A≥0 ∧ B>0 }
     local
          X,Y : integer
     begin
          X := A; Y := B;
     M:   while X ≥ Y loop
            X:=X-Y;
          end loop;
          Z:=X;
          { Z = (A MOD B) }
     end
```

Let $M(X,Y,z!) = wp(M, Z = z!)$ where M: LOOP; Z:= X
$= wp(LOOP, X = z!)$

$wp(LOOP, X = z!) = X \geq Y \wedge wp(X:=X-Y, wp(LOOP, X = Z!))$
$\vee\ X < Y \wedge X = Z!$

i.e.

$$M(X,Y,z!) = X \geq Y \wedge M(X-Y,Y,z!) \vee X < Y \wedge X = z!$$

Initialising M we get

$M(a,b,z!) = wp(A:=a;B:=b;INIT, M(x,y,z!))$
$= wp(MOD(a,b,z), z = z!).$

If we show

$$\vdash a \geq 0 \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a,b,z!) \qquad (1)$$

then

$$\vdash a \geq 0 \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow wp(MOD(a,b,z), z = z!)$$

i.e. $\vdash a \geq 0 \wedge b>0 \rightarrow wp(MOD, z = (a \text{ Mod } b))$ -- Substitution Capture Rule for wp (see 2.1 above)

We can prove (1) by Induction on a,

Summary

We wanted to show

$\vdash$ {a ≥ 0 ∧ b>0} MOD {z = (a Mod b)}   (2)

We associated with MOD a relation M(x,y,z!) such that

{a ≥ 0 ∧ b>0} MOD {z = (a Mod b)}

≡ a ≥ 0 ∧ b>0 ∧ z! = (a Mod b) → M(a,b,z!).

Instead of showing (2) we showed

$\vdash$ a ≥ 0 ∧ b>0 ∧ z! = (a Mod b) → M(a,b,z!).

More generally,

Given a function f, to show

$\vdash$ {x ε Dom(f)} PROG {z = f:x }

i.e. $\vdash$ x ε Dom(f) → wp(PROG, z = f:x)

we show,

$\vdash$ x ε Dom(f) ∧ z! = f(x) → wp(PROG, z = z!)

by considering the predicate wp(PROG, z = z!).

3. The Invariant

In establishing the partial correctness of a loop statement we generally use <u>an</u> invariant, there may be more than one. Finding a suitable invariant for an arbitrary program is not easy and so we are encouraged to develop an invariant when constructing a program. We will now consider what, in a sense, is <u>the</u> invariant of a loop, which would be an integral part of the program development. We return to the MOD example to elaborate first what we mean by an invariant, then we discuss what we mean by <u>the</u> invariant.

In the last section, in proving the MOD program correct, we showed by induction that

$\vdash$ a ≥ 0 ∧ b > 0 ∧ z! = (a Mod b) → M(a,b,z!)

that is

$\vdash$ a ≥ 0 ∧ b>0 ∧ z! = (a Mod b) → wp(MOD, z = z!),

since

M(a, b, z!) = wp(MOD, z = z!).

We now show how we can use an **invariant** to establish

$\vdash$ a ≥ 0 ∧ b>0 ∧ z! = (a Mod b) → wp(MOD, z = z!)

and so give an alternative way of proving this result.

Since wp(S, Q) = wlp(S, Q) ∧ wp(S, True), we show

1) $\vdash$ a ≥ 0 ∧ b > 0 ∧ z! = (a Mod b) → wlp(MOD, z = z!)

2) $\vdash$ a ≥ 0 ∧ b > 0 ∧ z! = (a Mod b) → wp(MOD, True)

where wlp(S, Q) is the weakest liberal precondition of S with respect to predicate Q.

Since MOD contains a loop we consider the general problem of showing

$\vdash$ P → wlp(L, Q)

where L: **while** B **loop** S

i.e. L = (**if** B **then** S;L **else** SKIP **fi**).

To show this we will use the following theorem, which is usually called the Invariant Theorem [3].

Invariant Theorem

Let L: **while** B **loop** S.

If 1) $\vdash$ (P ∧ ¬B) → Q

and 2) P is Invariant i.e. $\vdash$ (P ∧ B) → wlp(S, P)

then $\vdash$ P → wlp(L,Q)

Since $\vdash wp(S,Q) \rightarrow wlp(S,Q)$ and $\vdash wlp(S,Q) \wedge wp(S,True) \rightarrow wp(S,Q)$, we get

Corollary to Invariant Thm.

Let L : **while** B **loop** S

If 1) $\vdash P \wedge \neg B \rightarrow Q$

and 2) $\vdash P \wedge B \rightarrow wp(S, P)$

then $\vdash P \wedge wp(L, True) \rightarrow wp(L, Q)$.

4. 'Weakest' Invariant.

We have from logic that

$\vdash$ **false** $\wedge \neg B \rightarrow Q$

also, $\vdash$ **false** $\wedge B \rightarrow wlp(S, false)$

since $\vdash$ **false** $\rightarrow Q$ "false implies anything".

Thus the predicate **false** is an Invariant of the loop and in a sense **false** is the 'strongest invariant' (it implies every other invariant). The predicate, **false**, is no use as an invariant as it can never be initially established.

In addition, we can show that wp(L,Q), and similarly wlp(L,Q)), are also invariants because when

L: **while** B **loop** S i.e. L = **if** B **then** S;L **else** SKIP **end if**

we have

1) $wp(L,Q) \wedge \neg B \;=\; ((B \wedge wp(S, wp(L,Q)) \vee (\neg B \wedge Q)) \wedge \neg B$
$\;=\; \neg B \wedge Q$
$\rightarrow\; Q.$

2) $wp(L,Q) \wedge B \;=\; (B \wedge wp(S, wp(L,Q))$
$\rightarrow\; wp(S,wp(L,Q))$
$\rightarrow\; wlp(S,wp(L,Q))$

Since any invariant P implies wlp(L,Q), we get that wlp(L,Q) is the 'weakest invariant' .

In program verification to show

$\vdash Pre \rightarrow wp(Init;loop, Q)$

we need to find a predicate, P, which could be an invariant, such that

$\vdash Pre \rightarrow wp(Init, P)$

and $\vdash P \rightarrow wp(loop,Q)$

i.e. $\vdash P \rightarrow wlp(loop,Q) \wedge wp(loop,true)$.

As was mentioned above, the invariant **false** is not appropriate as from Pre we cannot establish **false** using Init, as for all programs S,

wp(S, **false**) = **false**.

The most appropriate invariant, and in a sense <u>the</u> invariant, if we could find it, would be wp(loop, Q) itself. Later we will consider the special case of finding wp(loop, z = z!) and how it relates to the fixpoint of the associated predicate of the loop.

Using an invariant we can show

$\vdash a \geq 0 \wedge b > 0 \wedge z! = (a \text{ Mod } b) \rightarrow wlp(MOD, z = z!)$

The following predicate

P: $x \geq 0 \wedge y > 0 \wedge z! = (x \text{ Mod } y)$

can be used as an invaraint.

(See Appendix 3).

5. Fixpoint and wp(loop,R)

We have associated with the loop: **while** B **loop** S, the relation wp(loop, z = z!).
Let L(x, z, z!) = wp(loop, z = z!).
The relation L(x, z, z!) satisfies the recursive equation,

$$L(x, z, z!) \;=\; B \wedge wp(S, L(x,z,z!)) \;\vee\; \neg B \wedge z = z!$$

We show that,

the (least) fixpoint of this equation = wp(loop, z = z!).

Let G be the functional (function on functions) such that

$$(G(L))(x, z, z!) \;=\; B \wedge wp(S, L(x, z, z!) \;\vee\; \neg B \wedge z = z!$$

The (least) fixpoint of G is given by

$$(U_{k>0}\, G^k(Z))(x,z,z!) \;=\; (\exists k)(k>0 \wedge (G^k{:}Z)(x,z,z!))$$

where Z is the empty or abort relation.

From Dijkstra's semantics of the loop and letting R: z = z! we have

$$wp(loop,R) \;=\; (\exists k)(k \geq 0 \wedge H_k(R))$$

where $H_0(R) = R \wedge \neg B$

$H_k(R) = B \wedge wp(S, H_{k-1}(R)) \vee H_0(R)$.

We can show by induction on k, (see Appendix 4)

$$G^{k+1}(Z) \;=\; H_k(R),\ k \geq 0$$

tf.

$$\begin{aligned}(U_{k>0}\, G^k(Z))(x,z,z!) &= (\exists k)(k \geq 0 \wedge H_k(R)) \\ &= wp(loop, R).\end{aligned}$$

5.1 Example:

Consider the Mod program,

```
MOD : x := a; y :=b;
loop: while x  ≥ y loop
          x := x-y;
      end ;
      z := x;
```

We have wp(loop; z := x, z = z!) = wp(loop, x = z!).
Let M(x,y,z!) = wp(loop, x = z!), therefore from the properties of wp,

$$M(x,y,z!) \;=\; x \geq y \wedge M(x-y, y, z!) \;\vee\; x < y \wedge x = z!$$

In this simple case we can find the fixpoint meaning of this equation by finding wp(loop, x = z!) explicitly.

5.2 Finding a fixpoint

$$wp(loop, x = z!) \;=\; (\exists k)(k \geq 0 \wedge wp(IF^k,\ x = z! \wedge x<y))$$

where IF: <u>if</u> $x \geq y \rightarrow x{:=}x-y$ <u>fi</u>

Let Q: $x = z! \wedge x<y$

$$wp(IF^0, Q) \;=\; x<y \wedge x = z!$$

$$\begin{aligned}wp(IF, Q) &= x \geq y \wedge x-y < y \wedge x-y = z! \\ &= y \leq x < 2y \wedge x-y = z!\end{aligned}$$

We can show by Induction, k > 0,

$$wp(IF^k, Q) \;=\; ky \leq x < (k+1)y \wedge x-ky = z!$$

therefore

$wp(loop, x = z!)$

$$= (\exists k)(k > 0 \;\wedge\; x = ky+z! \;\wedge\; ky \le x < (k+1)y) \;\vee\; x<y \;\wedge\; x = z!$$

$$= (x \ge y \wedge y > 0 \wedge z! = (x \text{ Mod } y)) \;\vee\; (x<y \wedge z! = x).$$

From this,

$$wp(MOD, z=z!) = wp(x:=a;y:=b, wp(loop, x=z!))$$

$$= (a \ge b \wedge b>0 \wedge z! = a \bmod b) \vee (a<b \wedge z! = x)$$

From Sect 2.4 above, we wanted to show

$$\vdash a \ge 0 \wedge b > 0 \wedge z! = a \bmod b \;\rightarrow\; wp(MOD, z = z!).$$

Having found wp(MOD, z = z!), we see that

$$\vdash a \ge 0 \wedge b > 0 \wedge z! = a \bmod b \;\rightarrow\; (a \ge b \wedge b > 0 \wedge z! = a \bmod b) \vee (a < b \wedge z! = x)$$

Since wp(loop, x=z!) is an invariant, we can regard the predicate

$$(x \ge y \wedge y > 0 \wedge z! = (x \text{ Mod } y)) \;\vee\; (x < y \wedge z! = x)$$

as the invariant of the loop as it also gives the fixpoint meaning of the associated relation of the loop.

6. Examples

For the purpose of comparison, we can summarise our approach to program verification by verifying two standard programs for factorial. One version is a ‘bottom up approach’, calculating fact(x) before calculating fact(x+1). The other version, a ‘top down approach’, is based on the recursive property of factorial, that fact(x+1) = (x+1) * fact(x).

Note:

Since we are using the symbol ‘!’ for the ‘out’ value of a variable, it precludes us from using N! for factorial N.

6.1 Example 1

We can calculate fact(N) = 1*2*...*N by

```
FACT: { N ≥ 0 }
      Init:     x:=1; y:=1;
      Loop: While x ≠ N loop
                x:=x+1;
                y:=y*x;
            end
            { y = fact(N) }
```

Let F(N,x,y,y!) = wp(Loop, y = y!).

The **in out** variable is x and the **out** variable is y. We can regard N as a constant.

We find that the associated predicate is

$$F(N, x, y, y!) = (x \ne N \rightarrow F(N, x+1, y*(x+1), y!) \wedge (x = N \rightarrow y = y!).$$

We look for a suitable invariant F1(N,x,y,y!) from F(N,x,y,y!) such that

1) $\vdash F1(N, x, y, y!) \wedge x \ne N \rightarrow F1(N, x+1, y*(x+1), y!)$

2) $\vdash F1(N, x, y, y!) \wedge x = N \rightarrow y = y!$

3) $\vdash N \ge 0 \wedge y! = fact(N) \;\rightarrow\; wp(Init, F1(N, x, y, y!))$
   i.e $\vdash N \ge 0 \wedge y! = fact(N) \;\rightarrow\; F1(N,1,1,y!).$

Let $F1(N,x,y,y!) = N \ge 0 \wedge y = fact(x) \wedge y! = fact(N).$

In particular, let the invariant be F1(N, x, y, fact(N)), i.e. let y! be fact(N).

Let INV: $N \geq 0 \wedge y = fact(x)$

We can show INV is an invariant by showing it satisfies the conditions

1) $\vdash N \geq 0 \wedge y = fact(x) \wedge x \neq N \rightarrow N \geq 0 \wedge y*(x+1) = fact(x+1)$

2) $\vdash N \geq 0 \wedge y = fact(x) \wedge x = N \rightarrow y = fact(N)$

3) $\vdash N \geq 0 \rightarrow N \geq 0 \wedge 1 = fact(1)$

From 1), 2) and 3) we conclude

$\vdash N \geq 0 \rightarrow wlp(FACT, y = fact(N))$.

Using the variant or bound function, N-x, we also can show the program halts,

tf. $\vdash N \geq 0 \rightarrow wp(FACT, y = fact(N)$.

6.2 Example 2

To calculate fact(N) we can also use the program

```
FACT:{ N ≥ 0 }
     Init:     x:=N; y:=1;
     Loop: while x ≠ 0 loop
               y:=y*x;
               x:=x-1;
           end
          { y = fact(N) }
```

Let $F(x, y, y!) = wp(Loop, y = y!)$,

$F(x, y, y!) = (x \neq 0 \rightarrow F(x-1, y*x, y!)) \wedge (x = 0 \rightarrow y = y!)$.

Show $\vdash N \geq 0 \rightarrow wp(FACT, y = fact(N))$.

In particular show

$\vdash N \geq 0 \wedge y! = fact(N) \rightarrow wp(Init;Loop, y = y!)$

but $wp(Init;Loop, y = y!) = wp(Init, F(x,y,y!))$

$= F(N,1,y!)$

i.e. show $\vdash N \geq 0 \rightarrow F(N,1, fact(N))$.

Find a suitable invariant F1(x,y,y!) from F(x,y,y!).

Let $F1(x,y,y!) = x \geq 0 \wedge y! = y*fact(x)$

In particular let

INV: $x \geq 0 \wedge fact(N) = y*fact(x)$

INV is a suitable invariant as

1) $\vdash x \geq 0 \wedge fact(N) = y*fact(x) \wedge x \neq 0 \rightarrow x-1 \geq 0 \wedge fact(N) = y*x*fact(x-1)$

2) $\vdash x \geq 0 \wedge fact(N) = y*fact(x) \wedge x = 0 \rightarrow y = fact(N)$

3) $\vdash N \geq 0 \rightarrow N \geq 0 \wedge fact(N) = 1*fact(N)$.

From 1), 2) and 3) we get

$\vdash N \geq 0 \rightarrow wlp(FACT, y = fact(N))$.

To show halting we can use the variant, x, as x decreases in the loop.

We get $\vdash N \geq 0 \rightarrow wp(FACT, y = fact(N))$.

To indicate the usefulness of an invariant we consider showing

$\vdash N \geq 0 \rightarrow F(N,1,fact(N))$

by induction on N.

We can write F(x,y,y!) as

$F(x,y,y!) = (x \neq 0 \wedge F(x\text{-}1,y*x,y!)) \vee (x = 0 \wedge y = y!)$. (1)

Proof (attempt)

N = 0

Since $x = 0 \wedge y = y! \rightarrow F(x,y,y!)$ we have in particular F(0,1,0!).

N > 0

Assume F(N-1,1,(N-1)!), show F(N,1,fact(N)).

From (1) we have in particular

$N > 0 \wedge F(N\text{-}1,N,fact(N)) \rightarrow F(N,1,fact(N))$

therefore ???...

(proof breaks down)

end Proof (attempt)

To make the Induction work we should attempt to prove something weaker

Show $\vdash N \geq 0 \rightarrow F(N, y, y*fact(N))$

therefore with y:=1 we get F(N, 1, fact(N)).

This weakening of our Induction assumption is analogous to the Dijkstra/Gries method of finding an invariant by weakening the post-condition by introducing a variable for a constant.

7. Summary and Conclusion

In this article we have presented an alternative approach to verifying imperative programs which like the standard approach also uses the notion of weakest precondition. Using the standard approach [Dijkstra], to show

$\vdash$ {P} F {Q}

one shows $\vdash P \rightarrow wp(F,Q)$.

Using conventions similar to those used in the verification of procedures [Gries] we can classify the variables in F into **'in'** variables and **'out'** variables.

Let $\underline{x}$ be the 'sequence' of **'in'** variables and
let $\underline{z}$ be the 'sequence' of **'out'** variables.

Some variables, the '**in out**' variables may appear in both 'sequences'.

Using this classification,

to show $\vdash \{P(\underline{x})\}\ F\ \{Q(\underline{x},\underline{z})\}$

we show $\vdash P(\underline{x}) \wedge Q[\underline{z} := \underline{z}!] \rightarrow wp(f, \underline{z} = \underline{z}!)$

where $Q[\underline{z} := \underline{z}!]$ is Q with all occurrences of $\underline{z}$ replaced by the new variables $\underline{z}!$ which denote the (unknown) final values of $\underline{z}$.

As an illustrative example we took the program

```
MOD: x := a; y := b;
      M:   while x ≥ y loop
             x := x-y;
           end
           z := x;
```

We associated with MOD the relation M(a,b,z!) where
M(x,y,z!) is defined by the recursive equation,

$M(x,y,z!) = x \geq y \wedge M(x\text{-}y,y,z!) \vee x < y \wedge x = z!$

To prove $\vdash \{a \geq 0 \wedge b>0\}$ MOD {z = a mod b}

we showed, by induction on a,

$\vdash a \geq 0 \wedge b>0 \wedge z! = a \bmod b \rightarrow M(a,b,z!)$

This proof involved using the properties of mod and the recursive definition for M(x,y,z!). While in this case it is straightforward, by inspection, to associate a relation with the imperative program, we showed in general, how we can, using the wp calculus, associate a relation with an imperative program.

Narrowing our focus to the loop statement, L: **while** B **loop** S, we can associate with the loop statement the predicate wp(L, $\underline{z}$=$\underline{z}$!). Let L($\underline{x}$, $\underline{z}$!) = wp(L, $\underline{z}$ = $\underline{z}$!)

From the properties of the wp calculus,

$L(\underline{x},\underline{z}!) \;=\; B \wedge \underline{z}=\underline{z}! \;\vee\; \neg B \wedge wp(S, L(\underline{x}, \underline{z}!))$

We can show that evaluating the least fixpoint of this equation is the same as evaluating wp(L, $\underline{z}$ = $\underline{z}$!). Since wp(L, $\underline{z}$ = $\underline{z}$!) is also an invariant of the loop, we can consider it <u>the</u> invariant of the loop.

In the MOD example we found that,
wp(loop_mod, x = z!) = $x \geq y \wedge y>0 \wedge z! = (x \bmod y) \;\vee\; x<y \wedge z! = x$
and we can consider this predicate as either being the invariant of the loop or the meaning of the loop.

To show $\vdash P \rightarrow wp(L, Q)$, we can avoid finding wp(L, Q) explicitly by using the Invariance Theorem. Having associated the relation L($\underline{x}$, $\underline{z}$!) with the loop L, we can take advantage of the invariance theorem to show $\vdash P \rightarrow L(\underline{x}, \underline{z}!)$.

To show $\vdash P \rightarrow L(\underline{x}, \underline{z}!)$

find an invariant Inv such that,

1) $Inv \wedge \neg B \;\rightarrow\; \underline{z} = \underline{z}!$
2) $Inv \wedge B \;\rightarrow\; wp(S, Inv)$
3) $P \;\rightarrow Inv$

Also we have to show,

4) $P \;\rightarrow\; Dom(L)$,
where Dom(L) determines the domain of L,
or in the wp calculus, Dom(L) = wp(L,True).

By enabling the verification of imperative programs to be carried out at the logic/functional level, this leads the way to formally developing imperative programs at the logic/functional level.

The translation from imperative programs to declarative programs is based directly on the weakest precondition predicate transformer. For ‘loop type’ programs we get tail recursive declarative programs but the technique can be applied to more general imperative programs that have, for example, recursive calls. Also, the technique can be adapted to non-deterministic programs constructed from the non-deterministic forms of the ‘if’ and ‘do’ statements.

The technique is quite mechanical and it may be possible to automate the process. As a first step, rather than translating a program directly to a declarative style, it could first be translated to a style of programming that would only have procedure and function calls. Traditionally, it was advised to remove the overhead of procedure and function calls, we are proposing the opposite in a effort to have a transition from structured ‘goto-less’ programs to declarative ‘assignment-less’ programs.

Appendix 1

Weakest Precondition and Weakest Liberal Precondition

If S is a (structured imperative) program and Q is a predicate then wp(S, Q) 'weakest precondition of S with respect to Q' describes the set of states such that is the program S starts in a state satisfying wp(S,Q) then the program S is guaranteed to halt in a state satisfying Q. The predicate, T (true), describes the set of all states and so wp(S, T) describes the set of states from which S halts.

The predicate, wlp(S,Q) 'weakest liberal precondition' is weaker than wp(S,Q)

i.e. $\vdash$ wp(S,Q) $\rightarrow$ wlp(S,Q)

Wlp(S,Q) describes the starting states of S such that, if S halts then S will halt in a state satisfying Q. If S is a program that we know halts, then wlp(S, Q) = wp(S,Q).

In general, wp(S, Q) = wlp(S, Q) $\wedge$ wp(S, T).

As an abbreviation, we say that "S establishes Q" if we know $\vdash$ wp(S,Q).

More generally, if $\vdash$ P $\rightarrow$ wp(S,Q) then we can say, that "S establishes Q from P".

Using weakest liberal precondition,

if $\vdash$ P $\rightarrow$ wlp(S,Q) then we can say, that "S partially establishes Q from P".

If S partially establishes Q, i.e. $\vdash$ wlp(S,Q), then we know that it is not the case that S will establish $\neg$ Q,

i.e. $\vdash$ wlp(S,Q) $\rightarrow$ $\neg$ wp(S, $\neg$ Q).

In other words, it we can establish wlp(S,Q) then at least we know that S won't give a wrong result but it may not give a result at all.

In Hoare triple form, [Gries] distinguishes partial correctness by giving two forms of the Hoare triple:

{P} S {Q} translates to P $\rightarrow$ wp(S, Q) -- total correctness

while

P {S} Q translates to P $\rightarrow$ wlp(S, Q) -- partial correctness.

The notation for wp(S,Q) [Dijkstra] has not been standardised and currently Dijkstra uses the 'curried' form wp.S.Q where '.' denotes function application.

Properties of wp

We consider the following properties related to wp, properties that are useful for program verification.

Law of the Excluded Miracle

For any program, S, wp(S, False) = False. It is impossible for a program to establish a contradictory predicate. In the case of wlp, If we are in a state satisfying wlp(S, False) then from above, we are not in a state satisfying wp(S, True), i.e. we are not a in state from which we are guaranteed to halt.

Distributivity of Conjunction

For any program, S, wp(S, Q $\wedge$ R) = wp(S, Q) $\wedge$ wp(S, R).

If S establishes a conjunction, Q $\wedge$ R, then S establishes the conjuncts separately.

Monotonicity

For any program S, if $\vdash$ Q $\rightarrow$ R then $\vdash$ wp(S, Q) $\rightarrow$ wp(S,R).

If we know that S establishes Q and we also know that Q implies R, i.e. R is weaker than Q, then we can conclude that S will also establish R.

Distributivity of Disjunction

For any program, S, $\vdash$ wp(S,R) $\vee$ wp(S,R) $\rightarrow$ wp(S, Q $\vee$ R).

If S is a deterministic program then equality holds, i.e. for deterministic S,

$$wp(S,R) \vee wp(S,R) = wp(S, Q \vee R).$$

In the context of this article, we assume that our programs are deterministic; for each input there is at most one ouput.

Equality of Programs

If for all predicates, Q we have that wp(S1, Q) = wp(S2, Q) then S1 = S2.

S1 and S2 may be different algoritms but if they cannot be distinguished by wp the two programs have the same meaning. For example, all ‘sorting’ programs are the same, in the sense that they have the same specification.

Calculating wp for basic statements

Assignment

wp(x := e, Q(x)) = Q(e)

replace all free occurrences of x in Q with e. We are are assuming that e is a well defined expression, otherwise we can resort to

$\vdash$ def(e) $\rightarrow$ wp(x := e, Q(x)) $\equiv$ Q(e), where def(e) is true when e is well defined.

Null Statement: skip

wp(skip, Q) = Q

Abort Statement: abort

wp(abort, Q) = False

It is impossible for the abort statement to establishes anything

Calculating wp for compound statements

Sequence of statements: S1;S2

wp(S1;S2, Q) = wp(S1, wp(S2, Q)).

Using the properties of wp, we can prove that the sequencing operator is associative.

For any predicate, Q,

$$\begin{aligned} wp(S1;(S2;S3), Q) &= wp(S1, wp(S2;S3, Q)) \\ &= wp(S1, wp(S2, wp(S3, Q))) \\ &= wp(S1;S2, wp(S3, Q)) \\ &= wp((S1;S2);S3), Q) \end{aligned}$$

tf. S1;(S2;S3) = (S1;S2);S3

We define

$$\begin{aligned} S^0 &= skip \\ S^{k+1} &= S;S^k \\ &= S^k;S \end{aligned}$$

Alernative statement: if

If we restrict oursleves to deterministic programs we can consider just the form
“if-then-else-fi”.

wp(**if** b **then** S1 **else** S2 **fi**, Q) = $(b \rightarrow wp(S1, Q)) \wedge (\neg b \rightarrow wp(S2, Q)$

$$= \quad (b \wedge wp(S1, Q)) \vee (\neg b \wedge wp(S2, Q)).$$

We also will make use of a 'one guard' if statement, **if** b $\rightarrow$ S **fi**.

wp(**if** b $\rightarrow$ S **fi**, Q) = b $\wedge$ wp(S, Q)

The iterative statement: while

We restrict ourselves to a determistic form, 'while - loop -end'

We can deifine wp(L, Q) where L is an iterative command but we may not be able to calculate it.

Let L : **while** b **loop** S **end**, then

$$wp(L, Q) \quad = \quad (\exists\, k: \mathbb{N}|: wp(IF^k, Q \wedge \neg b)) \ \ \underline{\text{where}}\ IF : \textbf{if}\ b \rightarrow S\ \textbf{fi}$$

$$= \quad (\exists k: \mathbb{N}\,|: k \geq 0 \wedge H_k(Q))$$

$$\underline{\text{where}} \quad H_0(R) = Q \wedge \neg b$$

$$H_k(R) = b \wedge wp(S, H_{k-1}(Q)) \vee H_0(Q).$$

Conclusion

A more complete treatment of weakest precondition given in [Gries], [Hehner] and also in [Snepscheut].

Appendix 2

Show

$\vdash a \geq 0 \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a,b,z!)$

<u>Proof</u> --by induction on, a

Given

$M(x,y,z!) = x \geq y \wedge M(x-y, y, z!) \vee x < y \wedge x = z!$ (1)

show

$\vdash a \geq 0 \wedge b > 0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a,b,z!)$.

Proof:

<u>Base case</u>**:** $0 \leq a < b$

From Assumption (1)

$\vdash a<b \rightarrow (z! = a \rightarrow M(a,b,z!))$ From Prop. Cal. $A \wedge B \rightarrow C = A \rightarrow (B \rightarrow C)$

From the properties of Mod

$0 \leq a<b \vdash a = a \text{ Mod } b$

tf.

$0 \leq a<b \vdash (z! = a \text{ Mod } b \rightarrow M(a,b,z!))$

tf.

$0 \leq a<b \vdash a \geq 0 \wedge b>0 \wedge z! = a \text{ Mod } b \rightarrow M(a,b,z!)$ since $0 \leq a<b \vdash a \geq 0 \wedge b>0$.

<u>Induction Step:</u>

Assume $0<b \leq a$

If true for $x<a$ show true for a

From (1)

$\vdash a \geq b \wedge M(a-b,b,z!) \rightarrow M(a,b,z!)$. (2)

Also $\vdash (a-b) < a$ since $b>0$

By Induction

$\vdash (a-b) \geq 0 \wedge b>0 \wedge z! = (a-b) \text{ Mod } b \rightarrow M(a-b,b,z!)$

From the properties of Mod, a Mod b = (a-b) Mod b, when $a \geq b$

i.e. $0<b \leq a \vdash z! = (a \text{ Mod } b) \equiv z! = (a-b) \text{ Mod } b$

tf. $0<b \leq a \vdash a \geq b \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a-b,b,z!)$

From (2)

$0<b \leq a \vdash a \geq b \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a,b,z!)$

tf. $0<b \leq a \vdash a \geq 0 \wedge b>0 \wedge z! = (a \text{ Mod } b) \rightarrow M(a,b,z!)$.

End Proof

Appendix 3.

```
MOD(in A,B:integer; out Z:integer) is
            { A≥0 ∧ B>0 }
            local
                  X,Y : integer
            begin
      MOD: X := A; Y := B;
            M:    while X ≥ Y loop
                    X:=X-Y;
                  end loop;
                  Z:=X;
                  { Z = (A MOD B) }
            end
```

Using an invariant we can show

$\vdash a \geq 0 \wedge b > 0 \wedge z! = (a \text{ Mod } b) \rightarrow \text{wlp}(MOD, z = z!)$

Let P: $x \geq 0 \wedge y > 0 \wedge z! = (x \text{ Mod } y)$.

Since MOD: X := A; Y := B; LOOP; Z := X;
we show first

$\vdash P \rightarrow \text{wlp}(LOOP, x = z!)$

We show P is an Invariant.

Show (i) $\vdash (P \wedge \neg B) \rightarrow x = z!$

(ii) $\vdash (P \wedge B) \rightarrow \text{wlp}(S, P)$.

i) $P \wedge \neg B = x \geq 0 \wedge y > 0 \wedge z! = (x \text{ Mod } y) \wedge x < y$

therefore from the properties of Mod we get

$\vdash (P \wedge \neg B) \rightarrow x = z!$.

ii) Since S: X:=X-Y we have

$\text{wlp}(S, P) = (x\text{-}y) \geq 0 \wedge y>0 \wedge z! = ((x\text{-}y) \text{ Mod } y)$

therefore from the properties of Mod we get

$\vdash (P \wedge B) \rightarrow \text{wlp}(S, P)$.

From i) and ii) and the Invariant Theorem we conclude

$\vdash P \rightarrow \text{wlp}(LOOP, x = z!)$

tf.

$\vdash \text{wp}(X := A; Y := B, P) \rightarrow \text{wp}(X := A; Y := B, \text{wlp}(loop, \text{wp}(z := x, z = z!)))$

i.e.

$\vdash a \geq 0 \wedge b > 0 \wedge z! = (a \text{ Mod } b) \rightarrow \text{wlp}(MOD, z = z!)$.

To show that the loop terminates, and therefore showing P is an appropriate invariant, we show

$\vdash x \geq 0 \wedge y > 0 \wedge z! = (x \text{ Mod } y) \rightarrow \text{wp}(loop, True)$.

It is sufficient to show

$\vdash (x \geq 0 \wedge y>0) \rightarrow \text{wp}(loop, True)$

We can do this by either

i) Use the Bound (Variant) function : x, which decreases in the loop,

or

ii) Find wp(loop, True) explicitly.

Let use choose the approach in ii)

Assume $x \geq 0 \wedge y>0$

show

$\vdash$ wp(loop, True)

wp(loop, True)	=	$(\exists k)(k \geq 0 \wedge wp(IF^k, x < y))$	where IF: **if** $x \geq y \;\rightarrow\; x{:=}x{-}y$ **fi**
$wp(IF^0, x<y)$	=	$x < y$	
wp(IF, x<y)	=	$x \geq y \wedge x\text{-}y < y$	
	=	$y \leq x < 2y$	
$wp(IF^2, x<y)$	=	$x \geq y \wedge (y \leq (x\text{-}y) < 2y)$	
	=	$2y \leq x < 3y$	

By Induction

$wp(IF^k, x < y) \quad = \quad ky \leq x < (k+1)y$

tf.

$wp(loop, True) \quad = \quad (\exists k)(k \geq 0 \wedge ky \leq x < (k+1)y).$

From Arithmetic

$x \geq 0 \wedge y>0 \vdash (\exists k)(k \geq 0 \wedge ky \leq x < (k+1)y)$

tf.

$x \geq 0 \wedge y>0 \vdash wp(loop, True).$

Appendix 4

Let G be the functional (function on functions) such that

$$(G(L))(x, z, z!) = B \wedge wp(S, L(x, z, z!)) \vee \neg B \wedge z = z!$$

Let

$$H_0(R) = R \wedge \neg B$$
$$H_k(R) = B \wedge wp(S, H_{k-1}(R)) \vee H_0(R).$$

Show by induction on k,

$$G^{k+1}(Z) = H_k(R)$$

Proof:

k=0, $G(Z) = \neg B \wedge R$
$= H_0(R)$

Induction step,
Assume

$$G^k(Z) = H_{k-1}(R)$$

show

$$G^{k+1}(Z) = H_k(R)$$

$$G^{k+1}(Z) = G(G^k(Z))$$
$$= B \wedge wp(S, G^k(Z)) \vee \neg B \wedge R$$
by Ind $= B \wedge wp(S, H_{k-1}(R)) \vee H_0(R)$
$$= H_k(R)$$

End Proof.

References

1. Dijkstra, E.W. A Discipline of Programming. Prentice-Hall, Englewood Cliffs, 1976.

2. Dijkstra, E.W. (ed) Formal Developments of Programs and Proofs. Addison-Wesley (University of Texas Year of Programming Series), 1990.

3. Gries, David.The Science of Programming Springer-Verlag, 1981, Ch. 11.

4. [Hehner] Hehner, Eric C.R. "*The Logic of Programming*"
Prentice-Hall International, 1984.

5. Knuth, Donald E. Structured Programming with goto statements. In: Knuth, D. Literate Programming. CSLI (University of Stanford) 1992, p 12.

6. Knuth, Donald E. Structured Programming with goto statements. In: Knuth, D. Literate Programming. CSLI (University of Stanford) 1992, p 19.

6. Manna, Zohar, "*Mathematical Theory of Computation*"
McGraw-Hill, 1974.

7. [Morgan] Morgan, Carroll "*Programming from Specifications*" 2nd edition
Prentice-Hall International, 1994.

8. [Snepscheut] Snepscheut, Jan L.A. van de "What Computing is all about"
Springer-Verlag 1993.

9. Spivey, J.M. "*The Z Notation*" 2nd Edition
Prentice-Hall International,1992.

10. Wordsworth, J.B. "*Software Development with Z*"
Addison-Wesley,1992.