Relating Z Specifications and Programs Through Hoare Logics

Antoni Diller

Abstract

A partial correctness specification is a triple $\{P\}\ \Gamma\ \{Q\}$ consisting of two predicates, namely P and Q, and a command Γ. A Hoare logic is a formal system for establishing various relations between such triples. In this paper it is shown how a partial correctness specification containing a dummy command can be derived from a Z schema. The now well-understood ways of developing a command from such a partial correctness specification can then be used to implement the original Z schema. This approach is better than its rivals because it uses well-known tried-and-tested methods.

Antoni Diller. Relating Z specifications and programs through Hoare logics. Research Report CSR–91–3, School of Computer Science, University of Birmingham, 1991.

Relating **Z** Specifications and Programs Through Hoare Logics

Antoni Diller
School of Computer Science
University of Birmingham
Edgbaston
Birmingham
B15 2TT

DillerAR@cs.bham.ac.uk

Abstract

A partial correctness specification is a triple $\{P\}\ \Gamma\ \{Q\}$ consisting of two predicates, namely P and Q, and a command Γ. A Hoare logic is a formal system for establishing various relations between such triples. In this paper it is shown how a partial correctness specification containing a dummy command can be derived from a **Z** schema. The now well-understood ways of developing a command from such a partial correctness specification can then be used to implement the original **Z** schema. This approach is better than its rivals because it uses well-known tried-and-tested methods.

Introduction

The use of formal specification methods in general and **Z** in particular is gaining ground in the programming community. Some evidence for this can be found in the growing number of textbooks available on the subject. Restricting my attention to **Z**, recent years have seen the publication of Ince (1988), Woodcock and Loomes (1988), Diller (1990), Potter, Sinclair and Till (1991) and Lightfoot (1991). Spivey (1989) is a comprehensive reference manual and the denotational semantics of **Z** is provided in Spivey (1988).

The general approach to constructing and developing a formal specification is now well established. First, a high-level specification is written employing mathematical data types the implementability of which is ignored for the time being. Then a lower-level specification is written which makes use of data types that are closer to the sorts of data type found in modern imperative programming languages (like Pascal and Modula). Various proof-obligations have to be discharged in order for the lower-level specification to be a correct refinement or reification of the higher-level specification.[1] This process of decomposition may have to be repeated several times. Eventually, a fairly low-level and concrete specification is arrived at.

Until recently, little if any formal work has been done on the problem of turning a low-level **Z** specification into a program. Spivey (1989), pp. 12–19, uses the reification approach, while Morgan (1990) has devised a special refinement calculus. In this paper I wish to suggest a different approach and that is to use a Hoare logic to bridge the gap between a low-level **Z** specification and the program that implements it. (Refining abstract specifications to concrete ones is still done as mentioned above.)

I think that this approach has much to recommend it. The use of Hoare logics is well understood. (See, for example, the books by Alagić and Arbib, (1978), Gries (1981), Backhouse (1986), Baber (1987), Gumb (1989), Dromey (1989) and Kaldewaij (1990).) Rather than devising new techniques to relate specifications and programs I think that tried-and-tested methods should be investigated first to see if they will work with **Z** specifications. In this paper I show that it is straightforward to relate a program to a **Z** specification by means of a few partial correctness specifications. The complications found in chapter 10 of Jones (1986) arising from the combination of VDM specifications and Hoare logics do not appear here. Sticking to a few simple conventions allows us to use the standard form of a Hoare logic.

A Small Specification

In order to illustrate how **Z** specifications can be related to programs by means of a Hoare logic I shall consider a very simple specification as my example. The state of this specification is given by the schema *Table*. This contains just one variable t and no predicates. The variable t can be thought of as an array of ten integers.

```
┌─ Table ──────────────────────────────
│ t: 1 .. 10 → Z
└──────────────────────────────────────
```

The schemas $\Delta Table$ and $\Xi Table$ are defined in the usual way.

$$\Delta Table \mathrel{\widehat{=}} Table \wedge Table',$$
$$\Xi Table \mathrel{\widehat{=}} \Delta Table \mid t' = t.$$

In the initial state all the elements of the array t' are set to 0.

$$InitTable' \mathrel{\widehat{=}} Table' \mid \forall i\colon 1 \mathinner{..} 10 \bullet t'(i) = 0.$$

Several operations will now be defined on this simple specification. The first, namely *Update*, just alters the value of one element in the array. The new value is represented by $v?$ and the number $p?$ indicates which array element is being updated.

[1]Details can be found in Diller (1990), chapter 13, and Spivey (1989), pp. 3ff.

```
── Update ──────────────────────────────
│ ΔTable
│ p?: N
│ v?: Z
├──────────
│ p? ∈ 1 .. 10
│ t' = t ⊕ {p? ↦ v?}
└───────────────────────────────────────
```

$$
\begin{array}{|l}
\textit{Update} \\
\hline
\Delta\,\textit{Table} \\
p?\colon \mathbf{N} \\
v?\colon \mathbf{Z} \\
\hline
p? \in 1\,..\,10 \\
t' = t \oplus \{p? \mapsto v?\} \\
\hline
\end{array}
$$

The second operation, namely *LookUp*, finds out what the value of a particular array element is. The position of the required element is given by $p?$ and the result is placed in $v!$

$$
\begin{array}{|l}
\textit{LookUp} \\
\hline
\Xi\,\textit{Table} \\
p?\colon \mathbf{N} \\
v!\colon \mathbf{Z} \\
\hline
p? \in 1\,..\,10 \\
v! = t(p?) \\
\hline
\end{array}
$$

The third operation, namely *Sum*, sums together the values of all the elements in the array and puts the result into the output variable $out!$

$$
\begin{array}{|l}
\textit{Sum} \\
\hline
\Xi\,\textit{Table} \\
out!\colon \mathbf{Z} \\
\hline
out! = \displaystyle\sum_{i=1}^{i=10} t(i) \\
\hline
\end{array}
$$

Summation is not part of standard **Z** as defined by Spivey (1989), but its meaning is so clear that there can be no harm in using it in specifications.

Obtaining Partial Correctness Specifications

Given a **Z** schema, like *Update*, it is easy to derive a partial correctness specification $\{P\}\ \Gamma\ \{Q\}$ from it that any command Γ must satisfy in order to be a correct implementation of the original **Z** schema. The Greek letter Γ here takes the place of the command that we have to find in order to implement the **Z** schema we started from.

The first step in deriving $\{P\}\ \Gamma\ \{Q\}$ is to calculate the precondition schema of *Update*. A precondition schema is obtained from a given schema by hiding all the after and output variables. Thus, *PreUpdate* is given as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{PreUpdate}} \\
\hline
t\colon 1\,..\,10 \to \mathbf{Z} \\
p?\colon \mathbf{N} \\
v?\colon \mathbf{Z} \\
\hline
\exists t'\colon 1\,..\,10 \to \mathbf{Z} \bullet \\
\qquad p? \in 1\,..\,10 \land t' = t \oplus \{p? \mapsto v?\} \\
\hline
\end{array}$$

This can be simplified to the following form:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{PreUpdate}} \\
\hline
t\colon 1\,..\,10 \to \mathbf{Z} \\
p?\colon \mathbf{N} \\
v?\colon \mathbf{Z} \\
\hline
p? \in 1\,..\,10 \\
\hline
\end{array}$$

The precondition P of the partial correctness specification that we are after is formed by constructing the conjunction of the predicate part of the precondition schema *PreUpdate* together with a predicate that consists of several identity statements. These identity statements provide the link between the **Z** specification and the actual program.[2] We require one identity statement for each variable declared in *PreUpdate*. (Because *PreUpdate* is obtained by hiding all after and output variables in *Update* all the variables it contains will be either before variables or input ones.) Each identity statement asserts the equality of a specification variable with a program variable. The program variables that are chosen must be entirely new, that is to say, they must be distinct from every variable occurring in our **Z** specification. In this paper I use the convention that program variables are written entirely in uppercase letters. Thus, the precondition that we are after is the predicate:

$$\mathit{PreUpdate} \land T = t \land P = p? \land V = v?,$$

where T, P and V are entirely new program variables.

Similarly, the postcondition of the partial correctness specification that we after consists of the conjunction of the predicate part of the schema *Update* together with a predicate that consists of several identity statements. If a before state variable t has been associated with a program variable T in the precondition part of our partial correctness specification, then the after state variable t' is associated with the *same* variable T in the postcondition. If an input variable $p?$ has been associated with a program variable P in the precondition, then it must be identified with the same program variable in the postcondition. Thus, the postcondition we are after is:

$$\mathit{Update} \land T = t' \land P = p? \land V = v?$$

[2]According to G. Polya, 'Setting up equations is like translating from one language into another.' (Quoted from Jones (1986), p. 23, where no reference is given.)

Putting all this together we obtain the partial correctness specification:

$$\vdash \{\mathit{PreUpdate} \wedge T = t \wedge P = p? \wedge V = v?\}\ \Gamma_1\ \{\mathit{Update} \wedge T = t' \wedge P = p? \wedge V = v?\}.$$

Here, Γ_1 represents the programming language commands that we are trying to find in order to implement *Update*. Replacing the schema names *PreUpdate* and *Update* by their respective predicates results in the partial correctness specification:

$$\begin{array}{l} \vdash \{p? \in 1\mathrel{..}10 \wedge T = t \wedge P = p? \wedge V = v?\} \\ \quad \Gamma_1 \\ \quad \{p? \in 1\mathrel{..}10 \wedge t' = t \oplus \{p? \mapsto v?\} \wedge T = t' \wedge P = p? \wedge V = v?\}. \end{array}$$

I shall call such a partial correctness specification *unsimplified* because clearly both the precondition and the postcondition can be considerably simplified by appropriate substitutions of equals for equals. Thus, any Γ_1 which satisfies the following partial correctness specification also satisfies the one just given:[3]

$$\begin{array}{l} \vdash \{p? \in 1\mathrel{..}10 \wedge \\ \quad T = t \wedge \\ \quad P = p? \wedge \\ \quad V = v?\} \\ \quad \Gamma_1 \\ \quad \{p? \in 1\mathrel{..}10 \wedge \\ \quad T = t \oplus \{P \mapsto V\} \wedge \\ \quad P = p? \wedge \\ \quad V = v?\}. \end{array}$$

Written in this form a suitable command Γ_1 is not difficult to find. It is $T := T \oplus \{P \mapsto V\}$ which is usually written in imperative programming languages as $T[P] := V$.[4]

Note that in order not to unduly complicate matters I have left type information in the partial correctness specification at an intuitive level. Thus, for example, it is not formally stated that T is an array variable.

We can go through the same steps with the schema *LookUp*. First, we form the precondition schema *PreLookUp*.

```
┌─ PreLookUp ──────────────────────────────
│ t: 1 .. 10 → Z
│ p?: N
├──────────
│ ∃t′: 1 .. 10 → Z; v!: Z •
│     p? ∈ 1 .. 10 ∧ v! = t(p?) ∧ t′ = t
└──────────────────────────────────────────
```

[3]To make these two partial correctness specifications identical several identity statements would have to added to the second one.

[4]Using the notation $T := T \oplus \{P \mapsto V\}$ for altering the component of an array allows us to use the usual Hoare logic axiom for assignment.

This simplifies to the following:

$$\begin{array}{|l}
\textit{PreLookUp} \\
\hline
t\colon 1 \mathinner{..} 10 \rightarrow \mathbf{Z} \\
p?\colon \mathbf{N} \\
\hline
p? \in 1 \mathinner{..} 10 \\
\hline
\end{array}$$

Thus, the required partial correctness specification is:

$$\vdash \{\textit{PreLookUp} \land T = t \land P = p?\}\ \Gamma_2\ \{\textit{LookUp} \land T = t' \land P = p? \land V = v!\}.$$

When written out in full this partial correctness specification is as follows:

$$\begin{array}{l}
\vdash \{p? \in 1 \mathinner{..} 10 \land T = t \land P = p?\} \\
\quad \Gamma_2 \\
\quad \{p? \in 1 \mathinner{..} 10 \land v! = t(p?) \land t' = t \land T = t' \land P = p? \land V = v!\}.
\end{array}$$

A suitable Γ_2 is $V := T[P]$.

Things are slightly more interesting when we come to the operation specified by the schema *Sum*. First, we work out the precondition schema *PreSum*:

$$\begin{array}{|l}
\textit{PreSum} \\
\hline
t\colon 1 \mathinner{..} 10 \rightarrow \mathbf{Z} \\
\hline
\exists t'\colon 1 \mathinner{..} 10 \rightarrow \mathbf{Z};\ \textit{out}!\colon \mathbf{Z} \bullet \\
\qquad \textit{out}! = \displaystyle\sum_{i=1}^{i=10} t(i) \land t' = t \\
\hline
\end{array}$$

This simplifies to the following:

$$\begin{array}{|l}
\textit{PreSum} \\
\hline
t\colon 1 \mathinner{..} 10 \rightarrow \mathbf{Z} \\
\hline
\end{array}$$

The partial correctness specification that we are after is, therefore:

$$\vdash \{\textit{PreSum} \land T = t\}\ \Gamma_3\ \{\textit{Sum} \land T = t' \land \textit{OUT} = \textit{out}!\}.$$

Here, Γ_3 represents the programming language commands that we are after which will implement the schema *Sum*. Writing out this partial correctness specification in full gives us:

$$\vdash \{T = t\}\ \Gamma_3\ \{t' = t \land \textit{out}! = \sum_{i=1}^{i=10} t(i) \land T = t' \land \textit{OUT} = \textit{out}!\} \tag{1}$$

It is not difficult to see that the following command is a suitable Γ_3:[5]

$$
\begin{array}{l}
\textbf{begin new } I;\\
\Delta\left\{\begin{array}{l}
\left.\begin{array}{l} OUT := T\,[1];\\ I := 1; \end{array}\right\}\Delta_1\\
\left.\begin{array}{l}\textbf{while } I \neq 10 \textbf{ do}\\ \Delta_3\left\{\begin{array}{l} I := I+1;\\ OUT := OUT + T\,[I]\end{array}\right.\end{array}\right\}\Delta_2
\end{array}\right.\\
\textbf{end}
\end{array}
$$

It is straightforward to show that this command is a suitable Γ_3 by means of the usual axioms and rules of a Hoare logic as found, for example, in Hoare (1969) and Hoare (1971). I will just give a sketch of the proof here.[6]

Clearly, any command Γ_3 which satisfies:

$$\vdash \{T = t\}\ \Gamma_3\ \{OUT = \sum_{i=1}^{i=10} T[i] \wedge T = t\} \tag{2}$$

also satisfies (1), so I will concentrate on (2). As I do not allow any specification variables, such as t, to occur in the commands that implement them (2) follows by specification conjunction from $\vdash \{T = t\}\ \Gamma_3\ \{T = t\}$ and:

$$\vdash \{\mathit{true}\}\ \Gamma_3\ \{OUT = \sum_{i=1}^{i=10} T[i]\} \tag{3}$$

So, I will concentrate my attention on establishing (3).

Using the assignment axiom twice and the sequencing rule it is easy to show that:

$$OUT = \sum_{i=1}^{i=I} T[i]$$

is an invariant of the command Δ_3. So, by precondition strengthening we get:

$$\vdash \{OUT = \sum_{i=1}^{i=I} T[i] \wedge I \neq 10\}\ \Delta_3\ \{OUT = \sum_{i=1}^{i=I} T[i]\}.$$

From this, by the rule for the **while**-loop, we can infer that:

$$\vdash \{OUT = \sum_{i=1}^{i=I} T[i]\}\ \Delta_2\ \{OUT = \sum_{i=1}^{i=I} T[i] \wedge I = 10\},$$

[5]When subscripting arrays I write, for example, $T[I]$, but when applying a function to an argument I use the notation $t(i)$. It would have been possible to use the same notation for both, but the practice employed is more common.

[6]The axioms and rules that I use in this proof sketch are summarised in the appendix to this paper.

and this is clearly the same as:

$$\vdash \{OUT = \sum_{i=1}^{i=I} T[i]\}\ \Delta_2\ \{OUT = \sum_{i=1}^{i=10} T[i]\}. \tag{4}$$

This completes that part of this proof dealing with the **while**-loop.

Using the assignment axiom twice and the sequencing rule it is straightforward to show that:

$$\vdash \{true\}\ \Delta_1\ \{I = 1 \wedge OUT = T[I]\}.$$

As the postcondition of this implies the precondition of (4) we have:

$$\vdash \{true\}\ \Delta\ \{OUT = \sum_{i=1}^{i=10} T[i]\}.$$

From this, by means of the block rule we can infer:

$$\vdash \{true\}\ \Gamma_3\ \{OUT = \sum_{i=1}^{i=10} T[i]\},$$

which is what I set out to prove initially. This completes this proof.

In real-life schemas are usually implemented by procedures (or functions). This could be done in this case. The resulting procedures might be:

$$\textbf{procedure}\ \mathit{UPDATE}\ (\textbf{in}\ P, V); \Gamma_1$$
$$\textbf{procedure}\ \mathit{LOOKUP}\ (\textbf{in}\ P; \textbf{out}\ V); \Gamma_2$$
$$\textbf{procedure}\ \mathit{SUM}\ (\textbf{out}\ OUT); \Gamma_3$$

In doing this the array variable T is being treated as a global variable.

General Applicability

In the previous section I used an extremely simple example of a **Z** specification to illustrate how it can be related to a program by means of a Hoare logic. In this section I want to show that the method is more widely applicable by applying it to one operation of the birthday book specification developed in chapter 1 of Spivey (1989). The operation in question is that of adding birthday information to a birthday book:

AddBirthday1

$\Delta \mathit{BirthdayBook1}$
$\mathit{name}?: \mathit{Name}$
$\mathit{date}?: \mathit{Date}$

$\forall i: 1 \mathinner{..} \mathit{hwm} \bullet \mathit{name}? \neq \mathit{names}(i)$
$\mathit{hwm}' = \mathit{hwm} + 1$
$\mathit{names}' = \mathit{names} \oplus \{\mathit{hwm}' \mapsto \mathit{names}?\}$
$\mathit{dates}' = \mathit{dates} \oplus \{\mathit{hwm}' \mapsto \mathit{date}?\}$

Using the method explained in the previous section it is straightforward to derive the following partial correctness specification from *AddBirthday1*:

$$
\begin{aligned}
\vdash\ & \{\forall i{:}\,1\,..\,hwm \bullet name? \neq names(i)\ \wedge \\
& \ NS = names \wedge DS = dates \wedge H = hwm \wedge N = name? \wedge D = date?\} \\
& \ \Gamma \\
& \{\forall i{:}\,1\,..\,hwm \bullet name? \neq names(i)\ \wedge \\
& \ hwm' = hwm + 1\ \wedge \\
& \ names' = names \oplus \{hwm' \mapsto names?\}\ \wedge \\
& \ dates' = dates \oplus \{hwm' \mapsto date?\}\ \wedge \\
& \ NS = names' \wedge DS = dates' \wedge H = hwm' \wedge N = name? \wedge D = date?\}.
\end{aligned}
$$

Here Γ holds the place for the command that we are after that implements the operation specified by *AddBirthday1*. This unsimplified partial correctness specification can be transformed and rewritten as follows:

$$
\begin{aligned}
\vdash\ & \{\forall i{:}\,1\,..\,H \bullet N \neq NS[i]\ \wedge \\
& \ H = hwm\ \wedge \\
& \ NS = names\ \wedge \\
& \ DS = dates\ \wedge \\
& \ N = name?\ \wedge \\
& \ D = date?\} \\
& \ \Gamma \\
& \{\forall i{:}\,1\,..\,H \bullet N \neq NS[i]\ \wedge \\
& \ H = hwm + 1\ \wedge \\
& \ NS = names \oplus \{H \mapsto N\}\ \wedge \\
& \ DS = dates \oplus \{H \mapsto D\}\ \wedge \\
& \ N = name?\ \wedge \\
& \ D = date?\}.
\end{aligned}
$$

Note that—as previously—I have left out some identity statements for clarity. When the partial correctness specification is written in this way the implementation almost jumps out at you from the page. It is not difficult to see that a suitable version of the command Γ is:

$$
\begin{aligned}
H &:= H + 1; \\
NS &:= NS \oplus \{H \mapsto N\}; \\
DS &:= DS \oplus \{H \mapsto D\}
\end{aligned}
$$

As already mentioned, the notation $NS[H] := N$ is more usual in programming languages than the form $NS := NS \oplus \{H \mapsto N\}$.

Conclusion

In this paper I have shown how a **Z** specification can be related to a programming language command by means of a Hoare logic. The conventions that I have employed are as follows:

(1) Program variables must be chosen that are different from all the variables used in the **Z** specification we are trying to implement.

(2) In an unsimplified partial correctness specification a conjunct containing both a program variable, X say, and a **Z** specification variable, x say, must be an identity statement of the form $X = x$ (or, of course, $x = X$).

(3) If a program variable, X say, is identified with a before **Z** specification variable, x say, in the precondition of an unsimplified partial correctness specification, then the same program variable must be identified with the corresponding after variable x' in the postcondition.

(4) If a program variable, P say, is identified with an output variable, $p?$ say, in the precondition of an unsimplified partial correctness specification, then the same program variable must be identified with the same input variable in the postcondition of the same partial correctness specification.

I have also shown how to derive a partial correctness specification containing a dummy command Γ from a **Z** schema specifying an operation employing the usual convention to distinguish between before and after variables.

Appendix: Hoare Logic Proof Rules Used

Assignment

$$\vdash \; \{P[E/V]\} \; V := E \; \{P\},$$

where $P[E/V]$ stands for the result of substituting E for all the free occurrences of V in P.

Sequencing

$$\frac{\vdash \; \{P_0\} \; \Gamma_1 \; \{P_1\} \quad \vdash \; \{P_1\} \; \Gamma_2 \; \{P_2\}}{\vdash \; \{P_0\} \; \Gamma_1 ; \Gamma_2 \; \{P_2\}} \; ;\text{-}\mathit{int}$$

While-loops

$$\frac{\vdash \; \{P \wedge S\} \; \Gamma \; \{P\}}{\vdash \; \{P\} \; \textbf{while} \; S \; \textbf{do} \; \Gamma \; \{P \wedge \neg S\}} \; \textbf{while}\text{-}\mathit{int}$$

Blocks and Local Variables The rule for blocks that I have used is from Hoare (1971) where, on p. 109, it is stated in this form:

$$\frac{\vdash \{P\}\ \Gamma[Y/X]\ \{Q\}}{\vdash \{P\}\ \textbf{begin new}\ X; \Gamma\ \textbf{end}\ \{Q\}}\ \textit{block-int}$$

where Y is not free in P or Q and it does not occur in Γ, unless Y is the same variable as X.

Bibliography

Alagić, S., and M.A. Arbib, (1978), *The Design of Well-Structured and Correct Programs*, Berlin, Springer-Verlag.

Baber, R.L., (1987), *The Spine of Software*: *Designing Provably Correct Software*: *Theory and Practice, or a Mathematical Introduction to the Semantics of Computer Programs*, Chichester, Wiley.

Backhouse, R.C., (1986), *Program Construction and Verification*, Hemel Hempstead, Prentice Hall.

Diller, A., (1990), **Z**:*An Introduction to Formal Methods*, Chichester, Wiley.

Dromey, R.G., (1989), *Program Derivation*: *The Development of Programs from Specifications*, Wokingham, Addison-Wesley.

Gries, D., (1981), *The Science of Programming*, Berlin, Springer-Verlag.

Gumb, R.D., (1989), *Programming Logics*: *An Introduction to Verification and Semantics*, Chichester, Wiley.

Hoare, C.A.R., (1969), "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, vol. 12, pp. 576–580 and 583.

Hoare, C.A.R., (1971), "Procedures and Parameters: An Axiomatic Approach", in E. Engeler (ed.), *Symposium on Semantics of Algorithmic Languages*, Berlin, Springer-Verlag, 1971, pp. 102–116.

Ince, D.C., (1988), *An Introduction to Discrete Mathematics and Formal System Specification*, Oxford, Oxford University Press.

Jones, C.B., (1986), *Systematic Software Development Using VDM*, Hemel Hempstead, Prentice Hall.

Kaldewaij, A., (1990), *Programming*: *The Derivation of Algorithms*, Hemel Hempstead, Prentice Hall.

Lightfoot, D., (1991), *Formal Specification Using* **Z**, Basingstoke, Macmillan.

Morgan, C., (1990), *Programming from Specifications*, Hemel Hempstead, Prentice Hall.

Morgan, C., K. Robinson and P. Gardiner, (1988), *On the Refinement Calculus*, Technical Monograph PRG–70, Oxford University Computing Laboratory.

Potter, B., J. Sinclair and D. Till, (1991), *An Introduction to Formal Specification and* **Z**, Hemel Hempstead, Prentice Hall.

Spivey, J.M., (1988), *Understanding* **Z**: *A Specification Language and its Formal Semantics*, Cambridge, Cambridge University Press.

Spivey, J.M., (1989), *The* **Z** *Notation*: *A Reference Manual*, Hemel Hempstead, Prentice Hall.

Woodcock, J.C.P., and M. Loomes, (1988), *Software Engineering Mathematics*: *Formal Methods Demystified*, London, Pitnam.