RAL-93-098

Science and Engineering Research Council

Rutherford Appleton Laboratory

Chilton DIDCOT Oxon OX11 0QX

RAL-93-098

Operation Semantics with Read and Write Frames

J C Bicarregui

December 1993

Operation Semantics with Read and Write Frames

Juan Bicarregui
Systems Engineering Division
S.E.R.C. Rutherford Appleton Laboratory, UK

Abstract

The read and write "externals" clauses in VDM implicit operation definitions play two distinct roles. On the one hand, they bind the free variables that appear in the preconditions and postconditions, whilst on the other hand, they can be understood to indicate the sets of state variables that an implementation of the operation can be permitted to "read" and "write". However, in the case of the read frame, existing semantic models for VDM give no formal interpretation to this latter role.

This paper investigates an extension to the denotational model of operations which captures this informal understanding of the read frame. It interprets the read frame via an extra constraint on the semantics of operations which formalises the idea that the behaviour of an operation should not depend on the variables outside the read frame. This semantics is shown to justify a new formalisation of the satisfiability obligation and to yield a property of non-interference between operations with sufficiently disjoint frames. However, some difficulties remain with this approach as a means of justifying algorithm refinement with read frames the resolution of which would require more substantial changes to the semantic model.

1 Introduction

The semantic role of the "read frame" in operation specifications is often underplayed. In VDM, implicit operation definitions have "read" and "write" clauses which serve two purposes. On the one hand, they bind the free variables that appear in the preconditions and postconditions, whilst on the other hand, they can be understood to indicate the largest sets of state variables that any implementation of the operation can be permitted to "read" and "write". Whilst for the write frame this role is formally interpreted by defining the meaning relation as if the postcondition were augmented with "rest unchanged" clauses; it is not the case that the read frame is interpreted as having any semantic meaning whatsoever. However, an informal understanding of what it means for an implementation to restrict its read accesses to a given set of state variables is clear.

If we are to maximise the benefit of formal notations and formal reasoning in the development of realistically sized systems, compositionality in design must be a primary concern. Modularisation of specifications, where the definition of the overall system state and operations is partitioned into modules with formal interfaces defined between them, can be the source of course-grained compositionality provided the modularisation mechanisms are sufficiently powerful to allow independent development of separate modules. However, this degree of independence between modules can lead to individual modules which are themselves complex systems.

Compositionality *within* modules, that is, independence in the development of operations (or parts of operations) that share a common state is also a possibility provided some means is given to define the areas of influence of individual operations. Just as the write frame determines the maximal set of state variables that the execution of an operation can influence, the read frame can define the maximal set of state variables by which the behaviour of an operation can be influenced. Properties of non-interference can then be defined for operations with sufficiently disjoint frames and these properties can be exploited to yield greater compositionality in the development of those operations. Thus read and write frames can be seen as a fine-grained counterpart to modularisation which provide for the compositional development of operations that share state within modules.

1.1 Background

Existing formalisations of operation decomposition in model-oriented specification pay little attention to the read frame, in particular, no formal interpretation has been given to the idea that an implicit operation specification might declare that only certain variables should be "read" by any implementation of it.

Jones [Jon87] gives a formalisation of operation decomposition in VDM incorporating some Hoare/Jones style rules describing valid decomposition steps which are justified against a denotational semantics based on relational algebra. This treatment deals in terms of the whole state and, whilst the write frame is interpreted as if the postcondition were augmented with "rest unchanged" clauses, that paper does not consider constraints on implementations imposed by the read frame. The same approach to read and write frames is also taken in the semantics for operations given in [VDM-SL].

In Z [Spi88], the distinction between read-only and read-write variables can be made in the declaration part of a schema through the use of Δ and Ξ in schema inclusion. The syntactic role of binding the variables in schema predicates is played by these declarations and an "open-world" interpretation is given to variables outside this scope (outside this frame anything can happen). This interpretation is useful for the incremental development of specifications as it provides for elegant interpretations of the schema combinators. However, no programming counterparts exist for some of these combinators and so the structure of the specification cannot necessarily be carried down into a structuring of the code. Hence such structuring is not a source of compositionality in development. Furthermore, no explicit treatment of the semantic role of the read frame is given as specifications are flattened before refinement occurs and hence read framing information is lost.

In the Refinement Calculus on the other hand [Mor88, Mor90], where the development of explicit operations is the primary concern, a "closed-world" assumption (outside this frame nothing changes) is made to enable convenient use of coding combinators. Here too, write frames are given an explicit treatment, but no attention is payed to the read frame.

Recent attempts to reconcile these two world views [Kin91, War93] have proposed some intermediate stances. In practice, a switch at some stage in the development from an open world view to a closed world view may suffice, but such a dichotomous approach would seem to be rather artificial.

In B's "Abstract Machines" [Abr93, ALNS91], the use of generalised substitutions as a means of specifying operations brings read and write information to the fore. Furthermore, the extensive facilities available for the structuring of specifications through various forms of machine composition, and the full-hiding and semi-hiding principles that arise from these structuring mechanisms, provide syntactic constraints to the state accesses that can be made by operations. In effect, this provides a form of read and write frames for the operations. Although these syntactic constraints do yield some degree of compositionality in development, no direct interpretation in the semantics is given and so the exact correspondence to the VDM style read and write frames discussed here is not clear.

[Bic93] gives a treatment of operation decomposition where the read and write frames are interpreted as "advanced information" about the state access that any eventual implementation of an operation can be permitted to make. This interpretation is formalised by a set of rules for algorithm refinement where each decomposition step respects the constraints imposed by the read and write frames. In effect, these rules give a definition of satisfaction that is a restriction of the usual refinement relation. However, no interpretation of the read frame in a relational semantics is given.

1.2 This paper

This paper develops a denotational semantics for operations where an interpretation is given to both read and write frames as constraints on the interpretation of operations. Thus it provides a semantic model which might form the foundation of a rule-based definition of algorithm refinement with read frames such as that given in [Bic93].

The next section recaps on the key definitions of [Jon87] and [Bic93] and section 3 gives some simple motivating examples. Section 4 gives a more formal presentation of this semantics and uses it justify a new definition of satisfiability and a property of non-interference. Finally, section 5 discusses some shortcomings of this formalisation and suggests some possible extensions to the work.

2 Recap

2.1 Relational semantics for operations

The interpretation of operations in [Jon87] is given by a pair of semantic functions. The first gives the termination set (of values of the state for which termination is guaranteed) and the second gives the set of possible state transitions (a relation on states).

$$\mathcal{T}: Stmt \to \mathcal{P}(\Sigma)$$
$$\mathcal{M}: Stmt \to \mathcal{P}(\Sigma \times \Sigma)$$

For any statement S, the domain of $\mathcal{M}(S)$ can be thought of as the set of states over which termination is possible and it is required that for all statements

$$\mathcal{T}(S) \subseteq \mathrm{dom}\,\mathcal{M}(S)$$

Refinement is defined in the usual way as a reduction in either non-determinism or undefinedness. For two statements S_1 and S_2, let

$$T_i = \mathcal{T}(S_i) \text{ and}$$
$$M_i = \mathcal{M}(S_i)$$

then S_1 being refined by S_2 is defined as follows[1]

$$S_1 \sqsubseteq S_2 \quad \triangleq \quad T_1 \subseteq T_2 \quad \wedge \quad M_1 \supseteq (T_1 \lhd M_2)$$

Proof rules for operation decomposition are given and justified against this semantics and compositionality and monotonicity are proven.

No treatment is given to parameters and results and the same simplification is taken here. Their treatment can be considered independently of the ideas developed here.

For the purposes of this paper the question of termination is not the focus of interest and thus $\mathcal{T}$ is not considered, rather it is assumed that

$$T = \mathrm{dom}\,M$$

and does not change during refinement.

2.2 Algorithm refinement with read and write frames

In [Bic93] it is argued that the syntactic and semantic roles of read and write frames can be usefully distinguished and an extra syntactic frame is proposed which binds the free variables that appear in the pre and postconditions and hence liberates the read and write frames to play their semantic role.

Thus an operation definition is composed of six components

$$\begin{array}{lrl} OpDef :: & frame & : \ \mathsf{map}\ Id\ \mathsf{to}\ Type \\ & invariant & : \ Exp \\ & reads & : \ Id\text{-}\mathsf{set} \\ & writes & : \ Id\text{-}\mathsf{set} \\ & pre & : \ Exp \\ & post & : \ Exp \end{array}$$

The first component is a declaration and binding of the variables appearing in the operation. The second records the pertinent clauses of the invariant. Third and fourth components give the "semantic" read and write frames, and the last two are pre and postcondition as usual.

Note that we do not insist on any relationship between the "externals" (the *reads* and *writes*) and the free variables of *pre* and *post*. Unlike the standard usage, here variables that can be read and written appear in both *reads* and *writes* clauses.

[1]Recall that $\lhd$ is domain restriction: for any set S and binary relation R over the same type, define the domain restriction of R to S to be the relation $S \lhd R \ \triangleq \ \{(s_1, s_2) \in R \mid s_1 \in S\}$

The paper gives a modified definition of satisfiability which takes account of the read and write frames and also defines some Hoare/Jones style rules for operation decomposition where extra constraints ensure that requirements encoded in the read and write frames are respected during refinement. Thus the satisfaction relation defined by these rules depends on the operation's read and write frames.

We recall the modified definition of satisfiability where the position of an extra universal quantifier inside the existential quantifier formalises the idea that the choice of new values for the writes must be valid whatever the value of the unread variables. For an operation *mk-OpDef* (F, I, R, W, P, Q) we have[2]

$$\forall \overleftarrow{\mathsf{R}} \cdot \exists \mathsf{W} \cdot \forall \overleftarrow{\mathsf{F}-\mathsf{R}} \cdot \overleftarrow{P} \wedge \overleftarrow{I} \;\Rightarrow\; \overleftarrow{Q}^{F-W} \wedge \overleftarrow{I}^{F-W}$$

3 Examples

We now present two simple examples which are sufficient to expose the key issues that will arise in the sequel.

Example 1

The first example has a state with two components and an invariant that restricts the state space to three possible values

$$\begin{aligned} \Sigma \;::\; & a : \mathbb{N} \\ & b : \mathbb{N} \end{aligned}$$

$$\begin{aligned} & \mathit{inv}\text{-}\Sigma(a, b) \;\triangleq \\ & \quad a \in \{0, 1\} \wedge \\ & \quad b \leq a \end{aligned}$$

Thus, the only possible states are

$$\{\mathit{mk}\text{-}\Sigma(0,0), \mathit{mk}\text{-}\Sigma(1,0), \mathit{mk}\text{-}\Sigma(1,1)\}$$

First consider the operation that simply assigns an arbitrary value to b (whilst respecting the invariant, naturally).

choose_b

ext rd b : $\mathbb{N}$

   wr b : $\mathbb{N}$

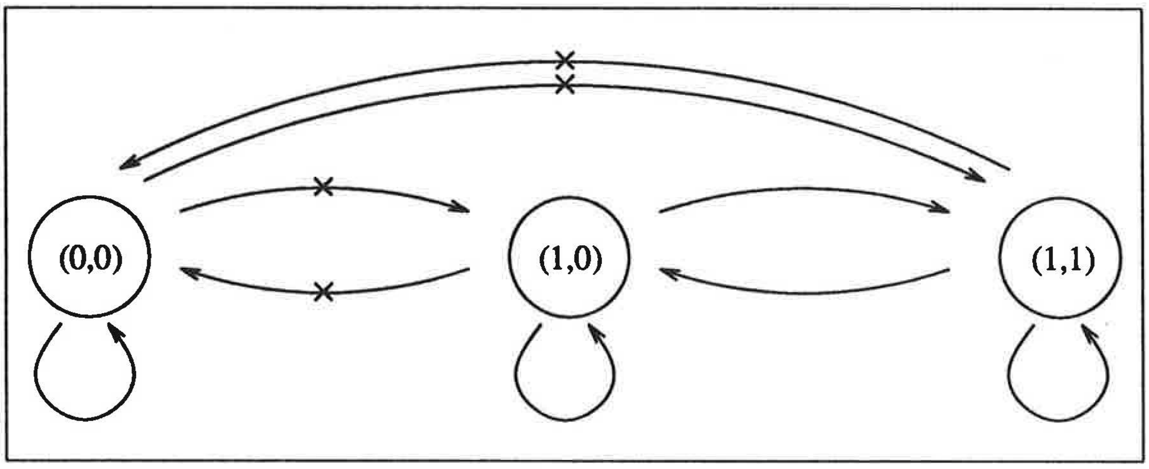
pre true

post true

[2] The notation is explained in detail in section 4.1.

Clearly, if both read and write frames are ignored, then the trivial predicates permit any transition between the three states that satisfies the invariant. Thus the interpretation is simply the trivial relation, $\Sigma \times \Sigma$.

It is a simple matter to interpret the write frame as a restriction of this relation by requiring that the a value cannot change. Thus, the four transitions that change the value of a are removed from the interpretation. In the following diagram these transitions are marked with an $\times$.



Now the central question addressed by this paper is whether it is possible to interpret the read frame by a similar constraint on the possible transitions? Or more precisely, whether those sets of transitions which *can* arise from code that respects the read frame can be characterised by some restriction of the usual semantics and the normal definition of satisfaction. This would in turn lead to a direct interpretation of the read frame in the semantics of the operation specification, rather than as a restriction on the definition of satisfaction.

In order to see how this might be possible, let us consider the validity of some possible implementations:

skip	- valid
$b := a$	- not valid, read frame not respected
$a := b$	- not valid: write frame not respected
$b := 1$	- not valid: invariant not respected
$b := 0$	- valid

It is in fact quite simple to list all the possible denotations of implementations which refine this specification. They are simply all the subsets of the relation which preserve its domain and are depicted in figure 1 (next page). Now trial and error will quickly establish that some of these interpretations can arise from code that respects the read frame whereas others cannot. These are indicated in the figure along with some example code that yields the valid interpretations.

It is claimed that any code yielding one of the interpretations marked as not valid cannot respect the read frame.

An inspection of these tentative implementations will quickly establish that the valid denotations are precisely those which do not contain the transition from $(1,0)$ to $(1,1)$. Thus the removal of this transition from the interpretation of the operation specification encodes the intention described by the read frame.

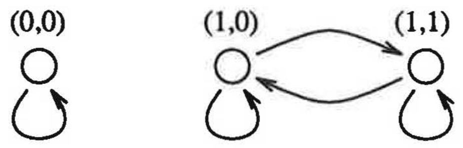
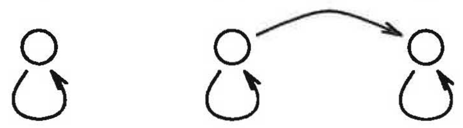
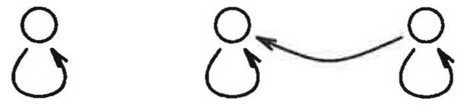
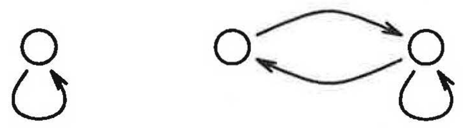
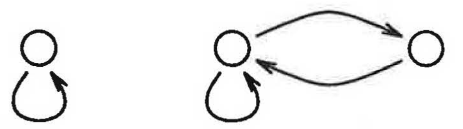
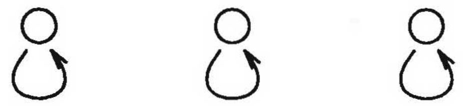
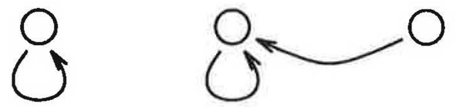
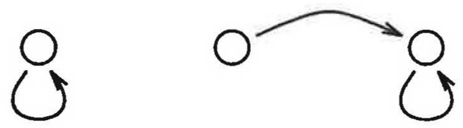
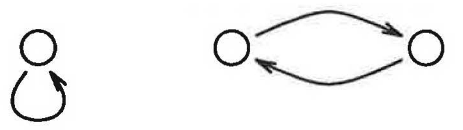
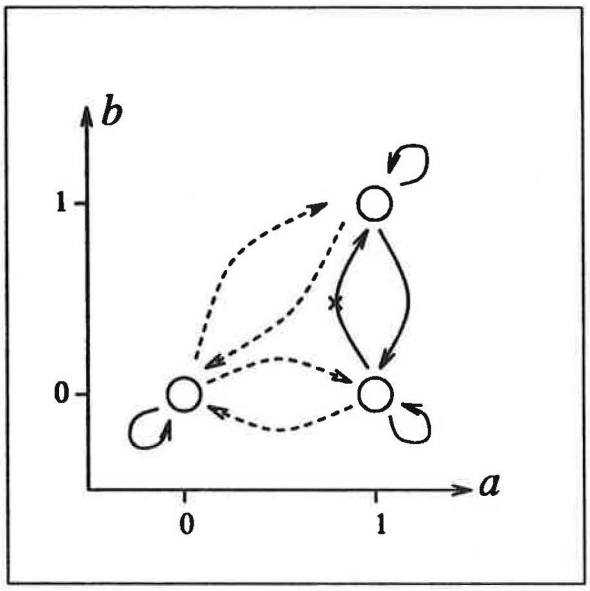
Interpretation	*Example code*	*Validity*
		No
		No
	`b := 0 [] skip`	Yes
		No
		No
	`skip`	Yes
	`b := 0`	Yes
		No
		No

Figure 1: All domain preserving subsets of the interpretation of *choose_b*.

To understand more fully why this is the case, it might be helpful to picture the state-space and state-transitions in two dimensions as follows:



The write frame constrains the transitions so that no movement in the a direction is possible. The read frame requirement removes one extra transition. This constraint on the semantic interpretation can be rationalised as follows:

> "Because, when $b = 0$, it is impossible, without looking at the a value, to establish whether the original state is $(0,0)$ or $(1,0)$, and because no change in the b value is possible in the case where the original state is $(0,0)$, therefore any such change in b must also be prohibited from state $(1,0)$."

With this constraint applied to the interpretation of operation specifications *at each stage in the decomposition process*, the usual refinement relation ensures that the read frame is respected.

Example 2

The second example is similar in nature to the first but has a slightly more complex state. The state has three components and an invariant which permits four valid states.

$$\begin{aligned} \Sigma \;::\; & a : \mathbb{N} \\ & b : \mathbb{N} \\ & c : \mathbb{N} \end{aligned}$$

$$\begin{aligned} & \mathit{inv}\text{-}\Sigma(a, b) \;\triangleq \\ & \quad a \in \{0, 1\} \land \\ & \quad b \leq a \land \\ & \quad c \leq b \end{aligned}$$

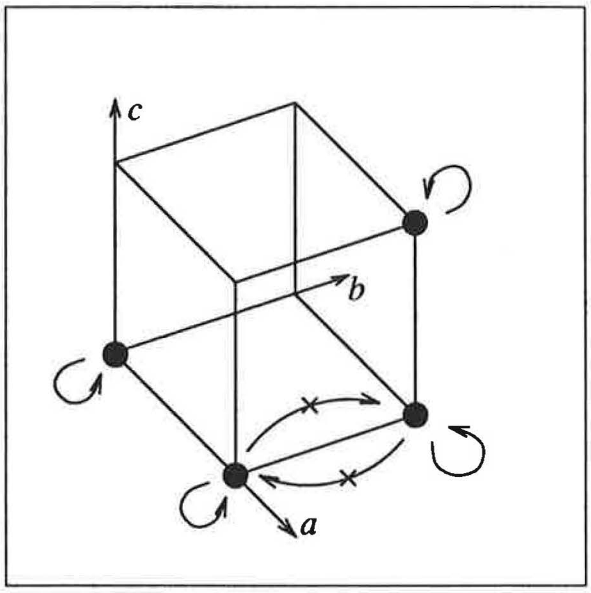
In this case the only possible states are

$$\{mk\text{-}\Sigma(0,0,0), mk\text{-}\Sigma(1,0,0), mk\text{-}\Sigma(1,1,0), mk\text{-}\Sigma(1,1,1)\}$$

Let us consider the operation that can read and write b only

```
$choose\_b$
ext rd $b$ : N
    wr $b$ : N
pre true
post true
```

Again ignoring the read frame in the first instance (but respecting the write frame), leads to six possible state transitions (including the four identity transitions or stuttering steps).



In this case, no state transitions (other than the stuttering steps) can arise from code that respects the read frame. Movement in each direction being prohibited by the existence of another state with the same unread values where that movement is not possible. Thus the only implementation that respects the read frame is skip.

(The reader may find it instructive to experiment further with this example by considering possible implementations for other combinations of variables in the read and write frames.)

4 Relational semantics with read and write frames

This section defines a denotational semantics for operations as relations on the state space (after the style of [Jon87]) but with an extra complexity introduced to formalise the interpretation of the read frame.

4.1 Notation

Some notation introduced in [Bic93] is summarised again here.

Hooking.

If S is any set of identifiers, say

$$S = \{x_a \mid a \in \alpha\}$$

then $\overleftarrow{S}$ is the set with each identifier in S distinguished in some way, with a $\leftharpoondown$ say. That is,

$$\overleftarrow{S} = \{\overleftarrow{x_a} \mid x_a \in S\}$$

More generally if we want to distinguish just some of the members of S, those in $S_1 \subseteq S$ say, then we write

$$\overleftarrow{S}^{S_1} \triangleq \{\overleftarrow{x_a} \mid x_a \in S_1\} \cup \{x_a \mid x_a \in S - S_1\}$$

Similarly, if E is an expression with free variables in S, written

$$E\colon Exp(S)$$

and $S_1 \subseteq S$, then

$$\overleftarrow{E}^{S_1} \triangleq E[\overleftarrow{\sigma_i}/\sigma_i]_{\sigma_i \in S_1}$$

Thus

$$\overleftarrow{E}^{S_1}\colon Exp(\overleftarrow{S}^{S_1})$$

Quantification.

Let F be a composite type

$$\begin{array}{lcl} F :: f_1 & : & T_1 \\ & \vdots & \\ f_n & : & T_n \end{array}$$

and let S be a subset of the field names of F

$$S = \{f_{S_1}, \ldots, f_{S_k}\} \subseteq \{f_1, \ldots, f_n\}$$

Let E be an expression with the field names in S appearing as free variables

$$E\colon Exp(f_{S_1}, \ldots, f_{S_k})$$

then we use the notations

$$\forall \mathsf{S} \cdot E \text{ and } \exists \mathsf{S} \cdot E$$

to stand for the quantifications over the free variables from S. For example:

$$\forall \mathsf{S} \cdot E \triangleq \forall v_1 \colon T_{S_1}, \ldots, v_k \colon T_{S_k} \cdot E[v_i/fs_i]_{i=1,\ldots,k}$$

where the v_i are fresh variables.
Similarly

$$\forall \overleftarrow{\mathsf{S}} \cdot \overleftarrow{E}$$

is a shorthand for the corresponding hooked formula.

Projection.

For an element of a composite type, define the projection onto a subset of the fields as the tuple composed of only those fields. For example

$$(a, b, c, d)\big\rfloor_{\{b,d\}} \triangleq (b, d)$$

Thus, for example, we have[3]

$$\overleftarrow{\sigma_i}^S = (\overleftarrow{\sigma_i}\big\rfloor_S, \sigma_i\big\rfloor_{F-S})$$

4.2 Semantic models

For a given operation, $mk\text{-}OpDef(F, I, R, W, P, Q)$, we define three relations on the state space giving interpretations for operations respecting none, one or both frames. Let Σ be the state space spanned by F, then we will define three meaning relations as follows

$\Sigma \xrightarrow{\circ} \Sigma \subseteq \Sigma \times \Sigma$ is the meaning relation not respecting either frame,

$\Sigma \xrightarrow{W} \Sigma \subseteq \Sigma \times \Sigma$ which respects, in addition, the write frame and

$\Sigma \xrightarrow{RW} \Sigma \subseteq \Sigma \times \Sigma$ which respects, in addition, the read frame.

For any $(\overleftarrow{\sigma}, \sigma) \in \Sigma \times \Sigma$ we write $\overleftarrow{\sigma} \xrightarrow{\circ} \sigma$ for $(\overleftarrow{\sigma}, \sigma) \in \Sigma \xrightarrow{\circ} \Sigma$, and similarly for $\overleftarrow{\sigma} \xrightarrow{W} \sigma$ and $\overleftarrow{\sigma} \xrightarrow{RW} \sigma$. Note that although single headed arrows are used in the relations above none of these relations are necessarily functions.

Thus we have

$$\Sigma \xrightarrow{\circ} \Sigma \triangleq \left\{(\overleftarrow{\sigma}, \sigma) \in \Sigma \times \Sigma \,\middle|\, Predicates(\overleftarrow{\sigma}, \sigma)\right\}$$

[3]Two minor syntactic liberties are taken with the presentation here. Firstly no account is taken of the order of the fields in the state and secondly the constructor function is omitted when convenient.

where

$$Predicates(\overleftarrow{\sigma},\sigma) \ \triangleq\ \overleftarrow{P} \wedge \overleftarrow{I} \ \Rightarrow\ Q \wedge I$$

It is a simple matter to extend this definition to incorporate the semantics of the write frame. The write frame is interpreted as if the postcondition were extended with "rest unchanged" clauses.

$$\Sigma \xrightarrow{\mathrm{W}} \Sigma \ \triangleq\ \left\{ (\overleftarrow{\sigma},\sigma) \in \Sigma \xrightarrow{\circ} \Sigma \,\middle|\, NoWrite((\overleftarrow{\sigma},\sigma), F-W) \right\}$$

where

$$NoWrite((\overleftarrow{\sigma},\sigma), F-W) \ \triangleq\ \sigma\big|_{\mathrm{F-W}} = \overleftarrow{\sigma}\big|_{\mathrm{F-W}}$$

The task now remaining, is to define an extra predicate that constrains the semantics as required by the read frame. However this predicate is not simply a predicate on individual transitions, but also depends on the whole set of transitions. Thus the intention in the following sections is to define the predicate *NoRead* taking a transition, a relation on the state space and a set of fields such that

$$\Sigma \xrightarrow{\mathrm{RW}} \Sigma \ \triangleq\ \left\{ (\overleftarrow{\sigma},\sigma) \in \Sigma \xrightarrow{\mathrm{W}} \Sigma \,\middle|\, NoRead((\overleftarrow{\sigma},\sigma), \Sigma \xrightarrow{\mathrm{W}} \Sigma, F-R) \right\}$$

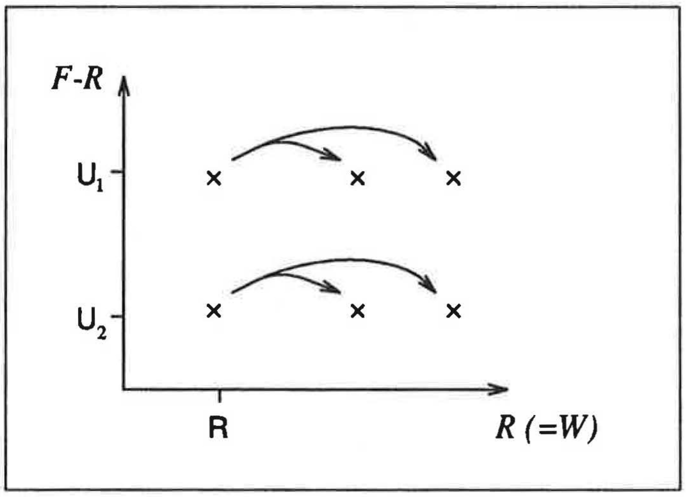
4.3 First Simplification

First consider the case where read and write frames are equal, $W = R \subseteq F$, and let $U = F-R$ be the unread part of the state.

From the examples above it is possible to describe informally the condition for respecting read frame as follows:

> "For any two states that differ only in the values of unread components, the possible final states must also differ only in the unread components (which cannot themselves change)".

This idea is depicted in the following diagram:



Movement in the $F-R$ direction is prohibited by the write frame, whereas the read frame constraint can be interpreted as saying that for any two starting states with the same R component, the same final values for the R component must be possible.

Thus, we require that $\Sigma \xrightarrow{RW} \Sigma$ be the largest subset of $\Sigma \xrightarrow{W} \Sigma$ such that

$$\forall \overleftarrow{R},\overleftarrow{U_1},\overleftarrow{U_2}\cdot\left\{R_1\,\middle|\,(\overleftarrow{R},\overleftarrow{U_1})\xrightarrow{W}(R_1,\overleftarrow{U_1})\right\}=\left\{R_2\,\middle|\,(\overleftarrow{R},\overleftarrow{U_2})\xrightarrow{W}(R_2,\overleftarrow{U_2})\right\}$$

which can be reexpressed as

$$\forall \overleftarrow{\sigma_1},\overleftarrow{\sigma_2}\cdot \overleftarrow{\sigma_1}\big|_{R}=\overleftarrow{\sigma_2}\big|_{R}\Rightarrow\left\{\sigma_1\big|_{R}\,\middle|\,\overleftarrow{\sigma_1}\xrightarrow{W}\overleftarrow{\sigma_1}^{F-R}\right\}=\left\{\sigma_2\big|_{R}\,\middle|\,\overleftarrow{\sigma_2}\xrightarrow{W}\overleftarrow{\sigma_2}^{F-R}\right\}$$

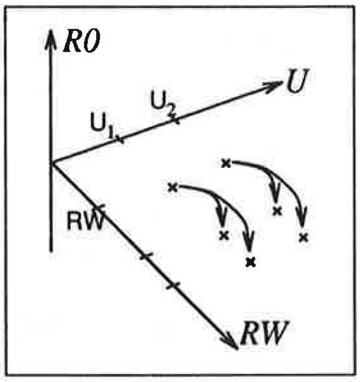
This can be ensured by defining the *NoRead* predicate as follows

$$\begin{aligned}&NoRead((\overleftarrow{\sigma}_1,\sigma_1),\Sigma\xrightarrow{W}\Sigma,F-R)\ \triangleq\\&\qquad\forall\overleftarrow{\sigma_2}\cdot\overleftarrow{\sigma_1}\big|_{R}=\overleftarrow{\sigma_2}\big|_{R}\ \Rightarrow\ \sigma_1\big|_{R}\in\left\{\sigma_2\big|_{R}\,\middle|\,\overleftarrow{\sigma_2}\xrightarrow{W}\overleftarrow{\sigma_2}^{F-R}\right\}\end{aligned}$$

4.4 First generalisation

The case where $W\subseteq R\subseteq F$ is really very similar. Simply, the R components are split in two parts. RO are the read-only components and RW are the read-write components. (U is still the unread part.)

The transitions can be pictured in three dimensions as follows:



Each transition lies in a horizontal plane, each horizontal plane satisfies the constraints of the previous section, but there is not necessarily any correspondence between planes.

Thus, we require that $\Sigma \xrightarrow{RW} \Sigma$ is the largest subset of $\Sigma \xrightarrow{W} \Sigma$ such that:

$$\begin{aligned}\forall \overleftarrow{RW},\overleftarrow{RO},\overleftarrow{U_1},\overleftarrow{U_2}\cdot&\left\{RW_1\,\middle|\,(\overleftarrow{RW},\overleftarrow{RO},\overleftarrow{U_1})\xrightarrow{W}(RW_1,\overleftarrow{RO},\overleftarrow{U_1})\right\}\\=&\left\{RW_2\,\middle|\,(\overleftarrow{RW},\overleftarrow{RO},\overleftarrow{U_2})\xrightarrow{W}(RW_2,\overleftarrow{RO},\overleftarrow{U_2})\right\}\end{aligned}$$

Which again can be reexpressed

$$\forall \overleftarrow{\sigma_1},\overleftarrow{\sigma_2}\cdot \overleftarrow{\sigma_1}\big|_{R}=\overleftarrow{\sigma_2}\big|_{R}\Rightarrow\left\{\sigma_1\big|_{W}\,\middle|\,\overleftarrow{\sigma_1}\xrightarrow{W}\overleftarrow{\sigma_1}^{F-W}\right\}=\left\{\sigma_2\big|_{W}\,\middle|\,\overleftarrow{\sigma_2}\xrightarrow{W}\overleftarrow{\sigma_2}^{F-W}\right\}$$

Note the judicious use of Rs and Ws in the subscripts and superscripts.

Thus, in this case we require

$$
\begin{aligned}
&NoRead((\overleftarrow{\sigma}_1, \sigma_1), \Sigma \xrightarrow{W} \Sigma, F-R) \ \triangleq \\
&\qquad \forall \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\big\lfloor_{R} = \overleftarrow{\sigma_2}\big\lfloor_{R} \ \Rightarrow \ \sigma_1\big\lfloor_{W} \in \left\{ \sigma_2\big\lfloor_{W} \,\middle|\, \overleftarrow{\sigma_2} \xrightarrow{W} \overleftarrow{\sigma_2}^{F-W} \right\}
\end{aligned}
$$

Note that, since

$$
\sigma_2\big\lfloor_{R-W} = \overleftarrow{\sigma_2}\big\lfloor_{R-W} = \overleftarrow{\sigma_1}\big\lfloor_{R-W} = \sigma_1\big\lfloor_{R-W}
$$

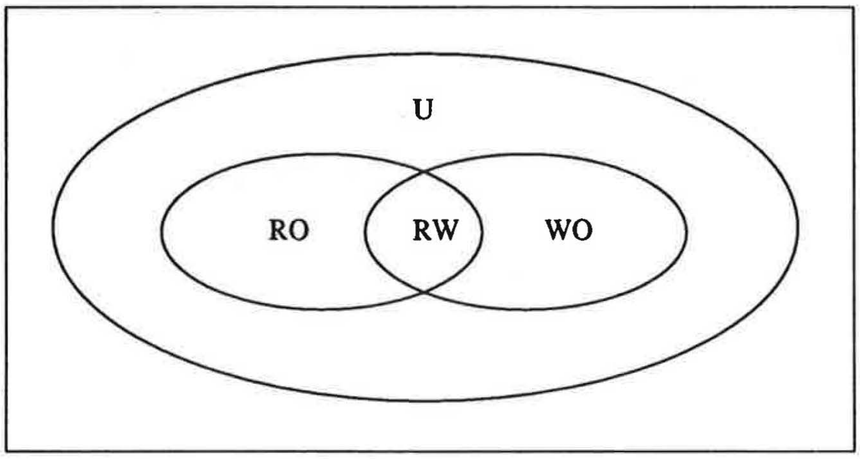
we can replace the Ws in the subscripts (and superscripts) by Rs, and so this predicate reduces to the one given for the case $R = W$.

4.5 Full generalisation

Before proceeding to consider the implications of this definition, we consider one more generalisation that goes beyond the normal usage in VDM. We remove the constraint that the reads must contain the writes.

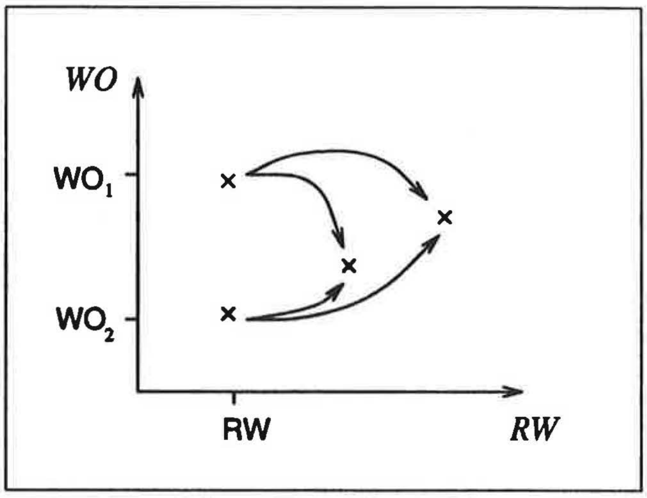
The interpretation that will be taken here for variables in the write frame but not the read is that they *must* be written with a value calculated from the reads. This "must-write" semantics is useful for formalising the initialisation of state and local variables and may also have some applications in security requirements (c.f. the Bell and LaPadula Model for confidentiality).

For this section we partition the state into four sets of fields. Now U is unread and unwritten, RO is read only, WO is write-only and RW is read-write:



In this general case, the transition diagram should best be drawn in four dimensions (one for each of U, RO, RW and WO). Fortunately, as no change is possible to either the RO or U components (that is there is no movement in the RO or U directions) each $RW \times WO$ plane can be pictured as a separate diagram. Planes separated by a change in U need satisfy the condition in section 4.3. Planes separated by a change in RO need have no correspondence.

Now within a $RW \times WO$ plane, two starting states which differ only in the WO component must have exactly the same possible final states - not just the same RW components:



Thus $\Sigma \xrightarrow{\mathsf{RW}} \Sigma$ is the largest subset of $\Sigma \xrightarrow{\mathsf{W}} \Sigma$ such that:

$$\begin{aligned}
&\forall \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{U}_1}, \overleftarrow{\mathsf{U}_2}, \overleftarrow{\mathsf{WO}_1}, \overleftarrow{\mathsf{WO}_2} \cdot \\
&\qquad \left\{ (\mathsf{WO}_1, \mathsf{RW}_1) \,\middle|\, (\overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{U}_1}, \overleftarrow{\mathsf{WO}_1}) \xrightarrow{\mathsf{W}} (\overleftarrow{\mathsf{RO}}, \mathsf{RW}_1, \overleftarrow{\mathsf{U}_1}, \mathsf{WO}_1) \right\} \\
&\quad = \left\{ (\mathsf{WO}_2, \mathsf{RW}_2) \,\middle|\, (\overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{U}_2}, \overleftarrow{\mathsf{WO}_2}) \xrightarrow{\mathsf{W}} (\overleftarrow{\mathsf{RO}}, \mathsf{RW}_2, \overleftarrow{\mathsf{U}_2}, \mathsf{WO}_2) \right\}
\end{aligned}$$

This says that for starting states with same read part (RO and RW) but different unread parts (U and WO): each transition keeps the unwritten (RO and U) components unchanged, whilst the possibilities for new values of the written components (WO and RW) must not depend on the old values – for the RW components, which are the same already this is ensured *a priori*, but for the WO component it means that any difference there may have been originally should be ignored. Thus the old values of WO components has no effect on the outcome.

Again this can be re-expressed as a quantification over states and, interestingly, this case also yields the same predicates as in the $W \subseteq R$ case.

However, as the standard usage in VDM does not consider write-only variables, the rest of this paper will concentrate on the case where $W \subseteq R$.

4.6 Satisfiability

In this section we sketch a justification for the modified definition of satisfiability defined in [Bic93].

The usual definition of satisfiability (not respecting the read frame) can be written

(1) $\quad \forall \overleftarrow{\mathsf{R}}, \overleftarrow{\mathsf{F}-\mathsf{R}} \cdot \exists \mathsf{W} \cdot \overleftarrow{P} \wedge \overleftarrow{I} \;\Rightarrow\; \overleftarrow{Q}^{F-W} \wedge \overleftarrow{I}^{F-W}$

The modified definition of satisfiability respecting frames is

(2) $\quad \forall \overleftarrow{\mathsf{R}} \cdot \exists \mathsf{W} \cdot \forall \overleftarrow{\mathsf{F}-\mathsf{R}} \cdot \overleftarrow{P} \wedge \overleftarrow{I} \;\Rightarrow\; \overleftarrow{Q}^{F-W} \wedge \overleftarrow{I}^{F-W}$

Under the present semantics we have the following extra property that

(3) $$\forall \overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}, \overleftarrow{\mathsf{U}_2} \cdot \left\{ \mathsf{RW}_1 \,\middle|\, (\overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}) \xrightarrow{\mathsf{w}} (\mathsf{RW}_1, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}) \right\}$$
$$= \left\{ \mathsf{RW}_2 \,\middle|\, (\overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_2}) \xrightarrow{\mathsf{w}} (\mathsf{RW}_2, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_2}) \right\}$$

where

$\mathsf{R} = \mathsf{RW} \cup \mathsf{RO}$,
$\mathsf{W} = \mathsf{RW}$, and
$\mathsf{F} - \mathsf{R} = \mathsf{U}$

In order to justify the new definition of satisfiability (2), we have to show that assuming (3), $(1) \Leftrightarrow (2)$.

The case $(2) \Rightarrow (1)$ is completely trivial. To see that $(1) \Rightarrow (2)$, we argue as follows. If for a particular $\overleftarrow{\mathsf{R}}$ and $\overleftarrow{\mathsf{U}_1}$ there is a suitable W, then by (3), the same W suffices for $\overleftarrow{\mathsf{R}}$ and $\overleftarrow{\mathsf{U}_2}$, which is (2).

4.7 Non-interference

One major motivation for the use of read frames in implicit operation definitions is to specify properties of non-interference that are to be achieved in implementations.

If we are to exploit the read and write frames fully, we would wish to have the following non-interference property for operations with sufficiently disjoint frames.

Theorem *Non-interference*

For two operations on the same context, F and I,

$$Op_i \triangleq (F, I, R_i, W_i, P_i, Q_i) \quad i \in \{1, 2\}$$

if the frames are sufficiently disjoint, that is if[4]

$$R_1 \cap W_2 = \{\} = R_2 \cap W_1$$

then we have

(1) $Op_1; Op_2 = Op_2; Op_1$

Furthermore, if each Op_i is refined by Op_i'

$$Op_i \sqsubseteq Op_i'$$

then we also have

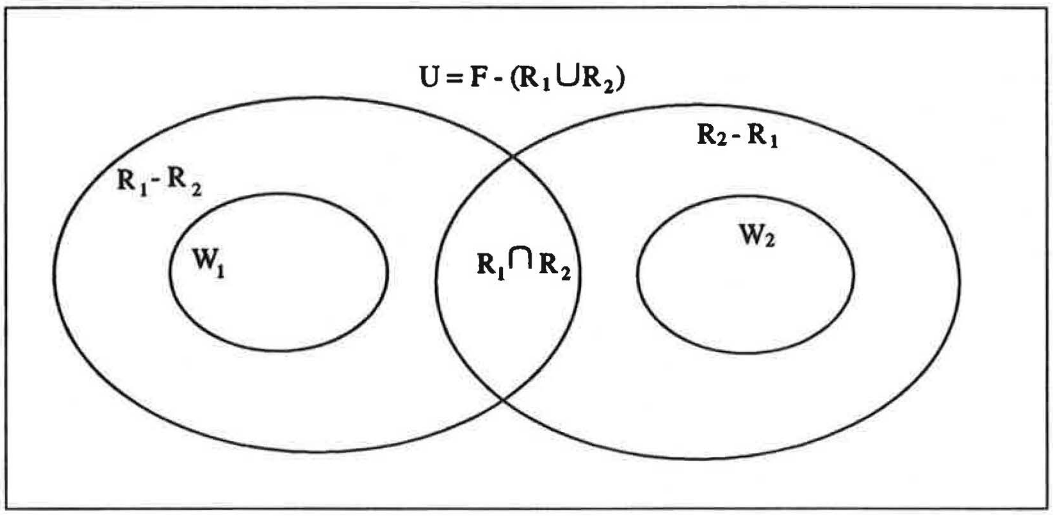
(2) $Op_1'; Op_2' = Op_2'; Op_1'$

[4]A similar condition called soft interference arises in process plant design. Each piece of equipment has a physical volume and a clearance volume. The latter being the space required for access and maintenance. Hard interference occurs if the physical volumes of two pieces of equipment intersect and soft interference if the physical volume of one piece intersects the clearance volume of the other. Two clearances are however allowed to intersect.

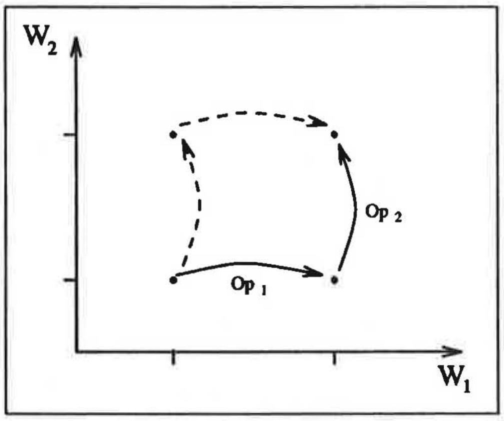
The proof of (1) is straightforward. We need to show that

$$[\![Op_1]\!]; [\![Op_2]\!] = [\![Op_2]\!]; [\![Op_1]\!]$$

To see this note that the state space is spanned by six components:



The only possible changes of state are in the plane spanned by the two write components. Let us picture this plane and consider an arbitrary transition from the sequential composition of the two operations $[\![Op_1]\!]; [\![Op_2]\!]$. Such a transition must clearly be the composition of one transition solely in the W_1 direction and another in the W_2 direction:



Clearly, the requirement is to show that there are necessarily transitions in the other order as depicted by the dotted arrows.

This follows since each write frames is contained within the unread component of the other operation and so the no-read property is sufficient to ensure the existence of the other transitions.

However, the second part of the theorem, that non-interference is preserved by refinement, raises some interesting issues which are discussed in the following section.

5 Discussion: Read Frames and Satisfaction

This paper has outlined a denotational semantics for operations that gives a formal interpretation to both the read and write frames and has shown how this formalisation of the read frame yields some new properties of satisfiability and non-interference. However it has still not addressed the interaction of this semantics with the satisfaction relation.

Let us return to the "central question" formulated in Section 3. Is it the case that with this modification to the interpretation of operations, the usual definition of refinement characterises the interpretations that can arise from implementations that respect the read frame?

To understand the question more fully, we need to consider the very basis for the semantic models and notion of refinement being used. In each of the three semantic models introduced in Section 4.2, the denotation of an operation as a relation on states is intended to characterise, via the refinement relation, the set of interpretations of its valid implementations. This is done by choosing as a representative of this set, the most general of its members. (In the lattice of the semantic domain under refinement, we take the meet of the set of refinements.) The refinement relation then determines whether any other interpretation is a member of this set, that is, whether it denotes a valid implementation.

In the case of the first two models, this approach is perfectly adequate since for any operation, the interpretation of any refinement is a refinement of the interpretation. (The set of interpretations of implementations is downwards closed in the lattice.) For the read-frame-respecting interpretations, the matter is not so simple.

When motivating the semantic interpretation of the read frame presented here, a crucial proviso was made in the claim that under this interpretation the usual definition of refinement would respect the read frame. It was stated in Section 3, that in order for the definition of refinement to respect the read frame, the constrained interpretation would have to be used *at each stage of the decomposition process*. Recall however, that the constraint itself depends on the read frame and thus we are assuming that some external mechanism is being employed to record this frame and apply *the same* constraint at later stages.

Unlike the $Predicates$ and $NoWrite$ conditions, the $NoRead$ condition does not automatically carry down to all refinements of an interpretation. That is, not all refinements of an interpretation satisfying the $NoRead$ predicate, necessarily satisfy the predicate themselves. Thus, it is not the case that the information in the frame is being captured in its entirety by the semantic model of the operation and hence the answer to the question posed above is *No*! It is not possible to characterise the interpretations of read-frame-respecting implementations as the refinements of a restricted interpretation of the operation specification. Thus, it is not possible to give a full interpretation to the read frame in the current semantic framework.

If we are to formalise the role of the read frame in refinement, we are left with two possibilities. Either we accept that the refinement relation must depend on the read frame and parameterise its definition accordingly, or we enrich the semantic model of operations to record more read information. Despite the fact that the read frame is part of the emerging standard for the VDM language [VDM-SL], neither of these two approaches has to date received any attention.

This latter approach is the subject of ongoing investigations by the author. The natural choice for a richer semantic model, which does indeed take full account of the information in the read frame, is to take as interpretation for an operation specification the set of relations on states that corresponds to its valid implementations. By filtering this set by the *NoRead* predicate, the role of the read frame as "advanced information" about the state accesses of valid implementations can be encoded in the interpretation of the operation specification. In effect, we have moved the definition of refinement into the semantics of the individual operations and refinement now becomes simply containment of sets of implementations. Although clearly we now have a more complex semantic model, this extra complexity is counterbalanced by a simplification of the refinement relation.

Of course, it should be remembered that such a semantics is not intended as a basis for the specifier's informal understanding, but rather as a basis for the justification of refinement rules with which it is possible to reason about frames. It is interesting to note that once again, syntactic devices for reasoning have developed well in advance of a formal understanding of their semantics. Although not an unusual occurrence this is perhaps surprising for a supposedly formal notation. As Wittgenstein concludes in his *Tractatus Logico-Philosophicus*[Wit22]

"Whereof one cannot speak, thereof one must be silent."

References

[Abr93] *Abstract Machines, Parts I, II and III.* J. R. Abrial. Unpublished, 1993.

[ALNS91] *The B Method.* J. R. Abrial, M. K. O. Lee, D. S. Neilson and P. N. Scharbach. VDM '91, Formal Software Development Methods (Tutorials), LNCS 552, Springer-Verlag (1991).

[Bic93] *Algorithm Refinement with Read and Write Frames*, J.C.Bicarregui, FME'93, LNCS 670, Springer-Verlag, 1993.

[Jon87] *VDM Proof Obligations and their Justification* C.B.Jones, Proceedings of the VDM '87 Symposium, LNCS 252, Springer-Verlag(1987).

[Kin91] S. King. Proceedings of VDM'90, pp. 164-188, LNCS 428, Springer-Verlag, 1991.

[Mor88] *On the Refinement Calculus.* C. Morgan, K. Robinson and P. Gardiner. Oxford University Technical Monograph, PRG-70, 1988.

[Mor90] *Programming from Specifications* C. Morgan, Prentice Hall, 1990.

[Spi88] J.M. Spivey. Understanding Z, Cambridge University Press, 1988.

[VDM-SL] VDM-SL, ISO Draft standard. ISO/IEC JTC1/SC22. N1346. April 1993. Also BSI IST/5/50. Draft 1st Decemeber 1992.

[War93] Nigel Ward, FME'93, LNCS 670, Springer-Verlag, 1993.

[Wit22] Wittgenstein, L. Tractatus Logico-Philosophicus, Routledge and Kegan Paul, First published 1922, translation by C.K. Ogden.