# Intra-Modular Structuring in Model-Oriented Specification: Expressing Non-interference with Read and Write Frames

Juan C. Bicarregui[1] [2]

Computing and Information Systems Department
Rutherford Appleton Laboratory
Chilton, Didcot
Oxon, OX11 OQX, U.K.
j.c.bicarregui@rl.ac.uk



[2]This work was done by external studies at the Department of Computer Science, University of Manchester, Oxford Road, Manchester, UK.

**Abstract**

Compositionality provides the key to managing complexity in software systems and thus should be sought at all levels in the design process. In the development of code from model-oriented specifications, compositionality can not only be achieved by the decomposition of system specifications into modules but may also arise within the development of individual modules.

This thesis considers the compositional development of operations that share state within modules. A key mechanism in the structuring of such developments lies in the definition of non-interference between sub-operations. The read and write frames in VDM implicit operations definitions can be understood as constraints on the read and write accesses which can be made by valid implementations. Their use as such permits reasoning about non-interference between operations and can thus be a source of compositionality within modules. However, existing semantic models of model-oriented specification and refinement do not provide a basis for such an interpretation of read and write frames.

Investigations are made into alternative denotational semantic models for operation specifications which do support read and write frames and a proof theory for reasoning about non-interference between sub-operations is founded upon them. The proposed semantic models provide progressively finer distinctions between implementations over and above the standard model. Thus new possibilities arise for discrimination between specifications that would otherwise be considered equivalent, whilst existing results remain valid.

# Contents

# Chapter 1

# Introduction

*"I do not know what is the highest degree of accuracy which is ever useful to an engineer – we shall be generous if we say ten significant figures. ... The number of primes less than 1,000,000,000 is 50,847,478: that is enough for an engineer, and he can be perfectly happy without the rest"*[1]

## 1.1 Compositionality and formal development

Compositionality is the key to managing complexity in systems design. In the development of complex systems, the ability to break down the task at hand, separately tackle each sub-task, and then bring the distinct solutions together in confidence that the whole will perform as required, provides the main apparatus with which it is possible to control complexity. The theory that relates the behaviour of the assembly to the behaviour of its component parts is the basis upon which sound compositional development is founded. Practical techniques for performing the calculations involved in the justification of such decompositions are prerequisite to the establishment of a viable engineering methodology.

In the design of software systems, it is the unambiguous specification of the appropriate system subcomponents that provides the source of compositionality. Such definitions should specify the syntactic form of the communication at the interfaces between components, as well as giving a semantic definition of the behaviours resulting from such communications. In a hierarchical or layered design, this amounts to the specification of the external behaviour of each component in terms of the external behaviour of its subcomponents.

A program text is a formal object. It is a term in a formal language with semantics defined by its interpretation by compiler and machine. When the behaviour required of a system component is also given a formal description, then the development process can be undertaken in a wholly formal world. That is not to say that every detail of the development need necessarily be backed up by a fully formal proof of its correctness,

[1]All the quotes at the start of chapters come from: G.H.Hardy, *A Mathematician's Apology.*

but rather, merely that the potential for such a proof exists. The designer is then free to provide as little or as much formality as desired, confident in the belief that further expansion of the formal details of the justification may be added as required. The use of a formal method is not, therefore, necessarily the use of a fully formal language and reasoning, rather it is the use of techniques which are understood to have the potential for expansion to full formality. The level of formality which it is appropriate to use in a particular situation will depend on factors such as the application area in which the system is to be employed, the cost of failure, or the degree of longevity required.

When deciding to undertake the formal development of software, the system designer must first consider which of the broad churches of formalism to adopt. In making this choice, the designer must consider the nature of the system to be developed, the architecture of the target system where one is mandated, and even the available skills of the personnel that will undertake the work. Among these criteria, the choice of methodology should also rely on the machinery of the formalism itself. For many purposes, it is the simplicity of the techniques for reasoning about the formal objects involved rather than the basis of the underlying mathematical foundations, that will be the relevant factor. By comparison, in the modelling of a physical system, the engineer will use mathematics appropriate to the domain rather than restrict the notation to the mathematics with the best understood foundation. The justification of the mathematical tools with which one reasons is a separate activity that need not necessarily be fully within the experience of the practising engineer. Indeed, it could be argued that it is the existence of application-oriented reasoning techniques that is the hallmark of a mature engineering discipline.

There is, of course, a direct correspondence between the development of reasoning techniques and their foundations. As in many areas of science, there is a continual push-pull relationship between a theory and its application. One can expect that in each particular area, the two will ultimately find a common expression. Surely, it is more than coincidence that, in mathematics, those theories that are older and have been repeatedly revised and reworked now seem intrinsically simpler than the newer ones which often appear more complex. This can be partly explained by the development of pedagogical methods for their explanation, but is also, in part, likely to be due to the convergence of theory and application. The number theorist G.H.Hardy contends that

> "there is no permanent place in the world for ugly mathematics"[Har40]

the same is probably true for any scientific or engineering discipline.

### 1.1.1 Decomposing specifications

The many benefits of structuring in software design will not be reiterated here. For the present discussion, it suffices to identify two distinct motivating principles for structuring which will be called *syntactic* and *semantic* structuring.

Syntactic structuring of specifications will be taken to mean structuring intended to facilitate understanding through the incremental presentation of specifications. That is, by the gradual introduction of the many features of a complex design, it is possible to "structure" the specification in such a way as to make it more readily comprehensible to the human reader. Such structuring is here called syntactic as the semantic interpretation of the specification is only given by first flattening out the structure. Here, subcomponents are not interpreted individually and their development cannot proceed independently of the development of rest of the system. Indeed, it is not necessarily intended that the structure of the specification should be reflected in the structure of the design. Experience suggests (e.g. [Hay93]) that this form of structuring is particularly useful when presenting a high level description of a system, abstraction ensuring that the description is sufficiently concise to allow simultaneous comprehension of the whole. Thus, although it is a useful tool for the presentation of specifications, particularly as a means to facilitate comprehension of the overall specification, this form of structuring does not in itself lead to compositional design.

Semantic structuring, on the other hand, is where the individual subcomponents can be fully understood without reference to the rest of the specification. Each component is given an interpretation in its own right and the semantics of the whole specification is composed of the semantics of the parts. The structure of the specification prescribes the intended structure of the system and the development of each component can be carried out in isolation from that of the rest of the system. In particular, it is possible to reason about the developments of the separate components individually and also about their composition, without knowledge of their separate developments. Thus, such structuring is the basis of compositionality in design and it is this form of structuring that is the primary interest in the present work.

In some senses, these two forms of structuring are the extreme positions on what is really a grey-scale of alternatives. An example of a form of structuring that lies somewhere between the two, is where the structuring of a specification does lead to a compositional semantics and hence to the possibility of independent validation of each component, but is not intended to indicate the structure of the implementation. It has been argued [Fit91] that this is a useful form of structuring for modularisation of model-oriented specifications.

### 1.1.2 Inter-modular and intra-modular structuring

The possibility for structuring developments arises at different levels in software design. In model-oriented specification, modularity is usually understood to mean coarse-grained structuring of a form corresponding to modules in code. In this sense, modules are essentially abstract data types with state: each module has a private state, the overall system state being the sum of these; the interface to the module is defined in terms of the signatures of a given set of operations; and no reference to the variables of internal states is permitted except via the operations in the signature of the module.

A modularisation mechanism such as this can provide syntactic or semantic structuring or indeed the middle position referred to above. No further discussion of compositionality arising from modularisation of specifications will be given here, other than to observe that the greater the degree of independence that is required between modules, the coarser the granularity of the modularisation is likely to be.

At a finer level of granularity, another form of structuring of specifications provides for the independent development of the components within each module. The most immediate source of intra-modular compositionality is in the development of the separate operations that manipulate a common state. Clearly, it is advantageous, indeed it is often taken for granted, that individual operations can be developed independently. Moreover, it is desirable that the development of each operation is amenable to a compositional approach. In practice, however, each operation is unlikely to be developed in isolation of the others. Typically, common sub-operations are identified and used in several operations. Thus, within individual modules there may be complex networks of dependencies between components.

### 1.1.3 Compositional development of operations

In the style of development of operations described for example in [Jon90a] or [Mor90], operation decomposition proceeds from an implicit definition of an operation towards an explicit one, whilst a justification that the latter is indeed an implementation of the former is constructed. This process can be incremental in nature, the intermediate stages being operations that have both implicit and explicit features. The implicit parts can be thought of as "holes" in the code that have yet to be filled in. These holes are the "specification statements" in [Mor88b] and are called "prescriptions" in [Mor87]. The specification of these holes is given by predicates which serve not only to document the development, but also provide the keystones that support the verification proof.

Compositionality requires the ability to develop each implicit part independently. If a strictly top-down approach is undertaken, then the context in which the hole occurs will contain all the information required for the predicates in its specification to be fully understood and hence for the hole to be satisfactorily "filled in". If, however, holes are to be developed independently of their context, then the definition of a hole requires more than the predicates in its specification. For example, information such as the declarations of the variables in scope, which are implicit in the context, must also be part of the hole's specification.

## 1.2 The Spec – Code gap

There are however a number of fundamental differences between specifications and code which hinder the smooth transition from implicit to explicit operations.

### 1.2.1 Open and closed worlds and the frame problem

The so called frame problem is an issue which arises in several branches of computer science as well as in logic and philosophy. What can be inferred about changes in one thing from changes in another? What assumptions can be made about entities which are not explicitly described?

For many purposes an open world assumption is convenient. Under this assumption nothing is assumed about things that are not mentioned. In the present context this means that if a specification defines certain aspects of the behaviour of a system component, then any behaviour that does not contradict that description is valid. Thus, if a specification says that certain variables must be assigned particular values, then any assignment is possible to other variables that do not appear in the predicates. That is "all can change unless otherwise stated". This is the interpretation that is made in the semantics of schemas in Z [Spi88] and gives an elegant mathematical explanation of the schema combinators in terms of logical manipulations of the component pieces of syntax.

On the other hand, in the closed world view it is assumed that variables not mentioned remain unchanged. This is a useful assumption for combining program segments, as variables not explicitly manipulated by a program can be assumed not to change. This assumption is used in [Mor90] and yields a natural interpretation of programming forms.

The dichotomy between these views has been noted in works that try to combine the two methodologies [Kin90], [War93]. Ward discusses the issue at some length and proposes a compromise between the two world views. Lamport also discusses the issue [AL90]. He observes that an open world assumption is essential for systems that interact with their environment whereas a closed world view is more useful for closed systems.

In VDM, the issue is addressed to some extent by the use of explicit read and write frames in operation specifications. Such frames provide "explicitness" by ensuring that all relevant variables are mentioned in the operation definition. However, although the "read" and "write" accesses to variables are syntactically obvious in code, in predicates they may not be so readily inferred. As a trivial example, the postcondition $x = y$ does not inform us whether it is $x$ that is to be written to the read value of $y$, or $y$ that is to be written to the read value of $x$, or indeed, whether both should be written to some third value. Furthermore, even before considering the possibility of data reification, the variables mentioned in a specification may not be the same as those referred to by the code that implements it. A trivial example of this might be a postcondition which asserts that some variable is unchanged by the predicate $x = \overleftarrow{x}$. Although syntactically it appears that $x$ is both read and written, in fact an implementation need not do either. Thus, explicitness requires that the read and write frames of an operation should be interpreted as giving "advanced notice" of the variables which are to be read and written by the code, rather than those which appear in the predicates of the specification.

### 1.2.2 Semantic structuring in predicates and code

Structuring mechanisms useful in the presentation of an operation specification do not necessarily provide a helpful means for structuring code. This dischord arises, in part, because the logical combinators for specifications do not have exact programming counterparts. Take, for example, the mainstays of the structuring of predicates, logical conjunction and disjunction. It is tempting to think that the correspondence that exists between disjunction and alternation is matched by a correspondence between conjunction and some form of parallel execution, but this latter correspondence only holds where the conjuncts are in some sense independent. This can be seen by comparing the specification, $x = 0 \wedge y = 0$, which could be implemented by a simultaneous assignment evaluating the two right hand sides and then assigning to the two left hand sides, with the logically equivalent assertion, $x = 0 \wedge y = x$, for which the corresponding implementation would be inappropriate, and for which the most obvious programming structure might be a sequence of two assignments. Clearly, the difference arises because in the latter case there is a dependency between the program variables.

Thus, if conjunction of predicates is to provide a source of compositionality in the development of operations, it will be necessary to be precise about the conditions under which it is possible to develop each conjunct separately. These conditions will depend on which programming operation is being used to compose the parts. As well as the possibilities of simultaneous assignment and sequencing mentioned above, another possible interpretation of conjunction in programming terms is as non-deterministic sequencing. Here, the components are executed in sequence, but the order in which they are executed is left unspecified. This interpretation has the advantage over sequencing that it permits the postponement of a design decision, and hence maintains a degree of abstraction in the description of an algorithm, while not requiring the degree of independence between components necessary for concurrent execution. These various "implementations" of conjunction lead to different requirements for ensuring non-interference between components. Non-interference can be ensured by constraining the read and write accesses to be made by the implementing code.

### 1.2.3 Non-Interference and Extensionality

Although read and write constraints on code can provide a sufficient condition for non-interference, conformance to such constraints is not a necessary condition since it is possible for code not conforming to given read and write constraints to exhibit behaviour as if it did. For example, whilst the code $x := \mathsf{if}\ y = 0\ \mathsf{then}\ 0\ \mathsf{else}\ 0$ would appear syntactically to read $y$, its extensional behaviour is identical to $x := 0$ which reads nothing. Similarly, $x := x$ writes $x$ but is equivalent to $\mathsf{skip}$ which does not.

For these two examples, it is hard to conceive a situation where the behaviour of the alternatives could be differentiated. However, another alternative for $x := 0$ is $\mathsf{if}\ y = 0\ \mathsf{then}\ x := y\ \mathsf{else}\ x := 0$, but these could be distinguished if they were executed in parallel with another operation that changed $y$. Thus, which equivalences between code fragments are appropriate depend on the atomicity of the possible interferences we wish to consider and this should be born in mind when defining a semantic model for operations. In particular, non-interference for shared variable parallelism, such as that considered in [Jon91], is not addressed here.

### 1.2.4 Non-interference and refinement

The above discussion freely uses the terms *specification* and *implementation*. The understanding behind these terms is that a specification describes a number of implementations, all of which exhibit the required behaviour. Correspondingly, the interpretation of a specification characterises, via the refinement relation, the set of interpretations of the valid implementations. This can be done by choosing as a representative of this set, the most general of its members. The refinement relation then determines whether any other interpretation is a member of this set, that is, whether it denotes a valid implementation.

In model-oriented specification, the usual definition of refinement is characterised by either a decrease in non-determinism or a decrease in undefinedness. The former states that the possible behaviours of the implementation must be contained within those of the specification. The latter, that the circumstances in which the implementation may not terminate, or for which we do not know the outcome, must be a subset of those for the specification. This defines a partial order on interpretations of specifications (by inclusion on sets of implementations).

This approach relies on the fact that for any operation, the interpretation of a refinement is a refinement of the interpretation. However, there is a slight obfuscation here which arises from the fact that not all interpretations arise from code. Apart from the possibility of non-computable values in the domain of interpretations, there are also other possible refinements which cannot arise from code. Say, for example, the programming language is such that all programs will be fully deterministic, then in order to specify that any one of several behaviours is acceptable, the specification may allow for non-determinism between them. This specification will be satisfied by a number of behaviours which are non-deterministic, however, no program with these interpretations can exist.

The constraint on implementations which leads to non-interference, and hence compositional development, is also not preserved by refinement. In particular, non-reading is not necessarily propagated by refinement since non-determinism present in a specification might be resolved in an implementation by reference to the value of some variable which should not be read. For example, the non-deterministic specification $x \in \{0, 1\}$, which does not exhibit any dependence on $y$, might be implemented by $x := \mathsf{if}\ y = 0\ \mathsf{then}\ 0\ \mathsf{else}\ 1$ where there is a clear dependence on the read value of $y$.

If we are to formalise the role of the read frame in refinement, we are left with two possibilities. Either we constrain the refinement relation in order that it respects the frame, or we enrich the semantic interpretation of operations to record the read information. Despite the fact that the read frame is part of the emerging standard for the VDM language [ISO93], neither of these two approaches has to date received any significant attention.

## 1.3 Interpreting the read and write frames

So we see that compositionality within the development of modules depends on semantic structuring in operation decomposition and this relies on properties of non-interference between their implementations. Non-interference can be couched in terms of the read and write accesses made by the code, however, there is not necessarily a direct correspondence between the variables referenced by the code and those in the predicates of the specification. Thus, if read and write frames are to be used to specify non-interference they must be interpreted as providing information about the variables accessed by the code, rather than those appearing in the specification. Verification that the read and write constraints imposed by an operation specification are satisfied by its implementation can be based upon either a constrained definition of refinement or an enriched semantic model of operations.

This thesis provides a framework for the compositional development of software within modules based upon the use of read and write frames to specify non-interference between code fragments. Proof rules for algorithm refinement are given which constrain implementations according to the read and write frames of the specification and thus define a refinement relation parameterised by the frames. The read and write requirements encoded in the proof rules are understood in terms of an informal model of a restriction on refinement to respect the frames. However, such a definition of refinement is undesirable as a semantic foundation, since conceptually, refinement should not depend on syntactic devices such as frames. The alternative approach is therefore preferred for the semantic model against which the soundness of the proof rules is demonstrated. An enriched semantics for operations is given which, in effect, moves the definition of frame respecting refinement into the semantics of the individual operations. Although this approach complicates the semantic model, the extra complexity is counterbalanced by a simplification in the definition of refinement.

## 1.4 Overview

The remaining chapters of this thesis are as follows:

**Chapter 2** An Example: A Model-oriented Development of a Communications Protocol.

This chapter presents an example development which though small, is complex enough to contain a few surprises. In particular, the interaction between the read and write frames, the invariant, and the precondition and postcondition is studied in detail.

**Chapter 3** A Proof Theory for Algorithm Refinement with Read and Write Frames.

This chapter proposes a proof theory which enables reasoning about read and write frames. In order to do this, it extends the underlying definition of operations and defines the idea of independence between parts of the state.

**Chapter 4** Interpreting the Read and Write Frames.

This chapter discusses semantic issues of the read frame. Normally ignored, the repercussions of read constraints on the semantics of operations requires careful analysis. Indeed, the chapter shows that with the usual model for operations, it is impossible to give a satisfactory semantics for algorithm refinement with read frames.

**Chapter 5** Operation Semantics with Read Frames.

This chapter defines a semantics addressing the issues discussed in Chapter 4. A number of separate issues are discussed and then brought together in the semantic model. In preparation for the next chapter, a slightly unconventional approach is adopted in the definition of this model.

**Chapter 6** Refinement with Read Frames.

This chapter enriches the semantics of Chapter 5 to ensure that read and write respecting behaviour is preserved by refinement. Although a more complex model is required, building on the approach taken in the previous chapter, it is a simple matter to define it.

**Chapter 7** Further Extensions to the Semantics.

This chapter discusses three issues which are not addressed by the semantics in Chapter 6. The first concerns the possibility of the introduction of write-only variables when the read frame is contracted during refinement. The second involes constraining the states which are visited during the execution of an operation, for example, ensuring that variables outside the write frame *never* change rather than simply revert to their original values. The third concerns the manipulation of the overall frame of operations, that is, expanding and contracting the variables in scope.

**Chapter 8** Conclusions.

This chapter draws some conclusions and discusses possibilities further research.

**Appendix A** Deferred Proofs.

This appendix presents some of the proofs from the main text in more detail.

The material from several chapters has been published elsewhere. In particular, the example in Chapter 2 formed the basis of a comparison of the VDM and B Abstract Machine notations published in [BR93], Chapter 3 was published in [Bic93] and Chapter 4 in [Bic94]. These three chapters remain basically in their published form and can be read fairly independently. Chapters 5, 6 and 7 expand the ideas in Chapters 3 and 4 and rely heavily on material in the earlier chapters.

# Chapter 2

# An Example: A Model-oriented Development of a Communications Protocol

*"It cannot be possible to prove mathematically that there will be an eclipse to-morrow, because eclipses, and other physical phenomena, do not form part of the abstract world of mathematics; and this, I suppose, all astronomers would admit when pressed, however many eclipses they may have predicted correctly."*

## 2.1 Introduction

In [BA92a][1], Bruns and Anderson describe a communications protocol in CCS with value-passing. A data model for the values is given that is, in effect, a model of the state of the device. This model is defined in terms of the type constructors usually found in model-oriented specification but without the use of invariants.

The part of the protocol that is described is a mechanism for manipulating a series of flags indicating the status of some shared-memory buffers. These flags are used to ensure that there is no "data-tearing" as multiple processors simultaneously read and write to the buffers. For the operations that update these flags, semaphores are used to ensure that each operation has uninterrupted access to the flags. Thus this behaviour can be described as a purely sequential system[2].

This chapter considers some alternative data-models for these status flags. In particular, attention is paid to the use of invariants in the data model and frames of reference in the operation definitions, neither of which are available in the data modelling language

[1] Now published as [BA94].

[2] Although the application concerns the simultaneous reading and writing of buffers, our interest in the system is not at this level. Rather we are concerned with the development of the subsystem manipulating the buffer status flags, where there is no parallelism.

of [BA92a]. It is argued that these features can play a key role in describing the system in a "natural" fashion and can thus help to deepen our understanding of the model. The example raises some stylistic questions concerning how much detail, that can be inferred from the invariant, should be left implicit in postconditions.

VDM is used for the development, and the role of the explicit frames of reference in operation definitions is examined in some detail. The interaction between read and write frames and invariant is studied, as is the manner by which the information in the frames prescribes the state accesses to be made by an implementation.

The chosen example is not the specification of a complete system, rather it is a description of a part of a system that could well form a module in the complete design. As such, the focus of attention in this chapter is with the development of a single module, and not the decomposition of a system into modules. Many issues are raised concerning the various constituent parts of such a module specification. This discussion motivates the particular interpretation of these specification components that will be developed in the subsequent chapters of this thesis.

The remainder of this first section is an informal description of the application and the desired protocol. The second section presents a formal specification of the system at a level of abstraction similar to the "abstract" description of [BA92a] and the third provides an alternative model of the system that makes it possible to write more useful framing information about the operations. The fourth section discusses some general points of methodology arising from the example and the final section summarises the issues raised in the chapter, indicating those topics which will be developed in what follows.

### 2.1.1 The Multiprocessor Shared-Memory Information Exchange (MSMIE)

MSMIE, Multiprocessor Shared-Memory Information Exchange, is a protocol that addresses intra-subsystem communications with

> "several features which make it ideally suited to inter-processor communications in distributed, microprocessor-based nuclear safety systems" [SBC$^+$89].

It has been used in the embedded software of Westinghouse nuclear systems designs.

The protocol uses multiple buffering to ensure that no "data-tearing" occurs as seperate processors communicate via some shared memory. That is, that data is never overwritten by one process whilst it is being read by another. One important requirement is that neither writing nor reading processes should have to wait for a buffer to become available, another is that recent information should be passed, via the buffers, from writers to readers. In the simplification considered in [BA92a] it is assumed that information is being passed from a single, writing, "slave" processor, to several, reading, "master" processors.

The information exchange is realised by a system with three buffers. Very roughly, at any time, one buffer is available for writing, one for reading and the third is either between a write and a read and hence contains the most recently written information, or between a read and a write and so is idle.

The status of each buffer is recorded by a flag which can take one of four values:

s - "assigned to slave." This buffer is reserved for writing, it may actually be being written at the moment or just marked as available for writing.

n - "newest." This buffer has just been written and contains the latest information. It is not being read at the moment.

m - "assigned to master." This buffer is being read by one or more processors.

i - "idle." This buffer is idle, not being read or written and not containing the latest data.

The names of the master processors that are currently reading are also stored in the state.

As mentioned earlier, neither the buffers themselves nor the slave and master processors that actually access the buffers in parallel are modelled here. This analysis concerns only the operations that modify the buffer status flags. In the system as a whole, these operations are protected by a system of semaphores which allow each operation uninterrupted access to the flags and thus their behaviour is purely sequential.

There are three operations:

*slave* This operation is executed when a write finishes. *slave* sets the status of the buffer that was being written to "newest" and so replaces any other buffer with this status.

*acquire* This is executed when a read begins. The new reader name (passed as a parameter) is added to the set of readers and status flags are updated as appropriate.

*release* Executed when a read ends, this operation removes a reader from the set and updates flags as appropriate.

The details of the behaviour of these operations are quite intricate and their precise description is left to the formal specification in the following section.

It should be noted however, that, as it stands, the protocol could have the undesirable property that information flow from slave to master is held up indefinitely by a series of overlapping reads requiring that a single buffer be continuously assigned to master. This possibility is ruled out in the original system [SBC+89] by timing constraints whereas [BA92a] suggests an improvement to the protocol (using a fourth buffer) that eliminates the possibility without recourse to timing arguments. A VDM specification of this improved protocol can be found in [BR93].

## 2.2 A VDM specification of MSMIE

The state in [BA92a] is defined as

> "a set of three pairs $(a, l)$ where $a$ is the buffer status, drawn from $\{\mathsf{i}, \mathsf{s}, \mathsf{n}, \mathsf{m}\}$, and $l$ is the buffer identification, drawn from $\{1, 2, 3\}$. The buffers are given as a set rather than a tuple to enable pattern matching rules in the description of the protocol."

The pattern matching rules do indeed give a concise description of the transitions of the system, in particular, the associative, commutative properties of sets are used to good effect in order to avoid much repetitive case analysis. However, the present author found that considerable effort was required to check that the patterns given were exhaustive and that the effects of overlaps between patterns were sensible. This difficulty is exacerbated by the fact that many of the states in the model are actually unreachable but no invariant on the state type is given to exclude them.

The specification given here makes an alternative choice of a sequence of three buffers for the state description. In addition, an invariant is used to exclude unwanted values from the state type.

### 2.2.1 The state

Possible values of the status flags are given via an enumerated type; the type of the names of master (reading) processors is deferred.

types

$$Status = \{\mathsf{s}, \mathsf{m}, \mathsf{n}, \mathsf{i}\}$$

$$MName = \mathsf{token}$$

The state is composed of three buffer status flags and a set of the names of the currently reading masters. The invariant captures the fact that only certain states are reachable by the operations. It gives restrictions on the possible combinations of status flags, namely that there is always exactly one buffer assigned to the writing slave; there is at most one currently being read and at most one with newest data that is not being read; and requires that the set of readers is empty precisely when there is no buffer being read. The initial state assigns one buffer to the slave and records that the other two buffers are idle.

$$
\begin{array}{l}
\mathsf{state}\ \Sigma\ \mathsf{of} \\
\qquad b\ :\ Status^* \\
\qquad ms\ :\ MName\text{-}\mathsf{set} \\
\mathsf{inv}\ mk\text{-}\Sigma(b, ms)\ \ \triangleq\ \ \begin{array}[t]{l} \mathsf{len}\ b = 3 \wedge \\ count(\mathsf{s}, b) = 1 \wedge \\ count(\mathsf{m}, b) \in \{0, 1\} \wedge \\ count(\mathsf{n}, b) \in \{0, 1\} \wedge \\ (count(\mathsf{m}, b) = 0 \ \Leftrightarrow\ ms = \{\,\}) \end{array} \\
\mathsf{init}\ mk\text{-}\Sigma(b, ms)\ \ \triangleq\ \ b = [\mathsf{s}, \mathsf{i}, \mathsf{i}] \wedge ms = \{\,\} \\
\mathsf{end}
\end{array}
$$

where[3]

$$count : Status \times Status^* \to \mathbb{N}$$

$$count(s, ss)\ \ \triangleq\ \ \mathsf{card}\,(ss \rhd \{s\})$$

### A validation condition on the state

We observe that only four combinations of buffers are allowed by the invariant:

$$\forall mk\text{-}\Sigma(b, ms){:}\Sigma \cdot \{b(1), b(2), b(3)\}_m \in \{\{\mathsf{s,i,i}\}_m, \{\mathsf{s,i,n}\}_m, \{\mathsf{s,i,m}\}_m, \{\mathsf{s,n,m}\}_m\}$$

where we have used $\{\ldots\}_m$ as a notation for bags (multisets), for example $\{\mathsf{s,i,i}\}_m$ is the bag containing one 's' and two 'i's.

Note that the invariant has captured, and brought to the fore, properties that would otherwise have to be deduced by looking in detail at the definitions of the operations. It makes it possible to build quickly our intuition of the workings of the specified machine. We know immediately that there is always one buffer reserved for writing, at most one being read, and at most one with newest data not being read.

## 2.2.2 The operations

### Slave

The first operation, *slave*, is executed when a write completes. It reassigns the status of the buffer just written, previously s, to n, thus replacing any other n buffer. It also non-deterministically chooses another available buffer which is to be the new buffer reserved for writing and assigns to it status s.

[3] Here, range restriction is used on sequences, viewing them as maps from natural numbers to elements.

$slave$ ()
ext wr $b$ : $Status^*$
pre true
post $\forall i \in \{1, 2, 3\} \cdot$
$(\overleftarrow{b}(i) = \mathsf{s} \;\Rightarrow\; b(i) = \mathsf{n}) \wedge$
$(\overleftarrow{b}(i) = \mathsf{m} \;\Rightarrow\; b(i) = \mathsf{m})$

The postcondition may, at first sight, seem to be too liberal: what should happen to any buffer that had status n or i? However, in conjunction with the invariant and the frame, it ensures that no other n buffer remains, that exactly one new s buffer is chosen, and that no new m buffers are added. Thus, for example, we can write the following validation property for *slave* which can be proved in order to increase confidence in the correctness of the postcondition:

$$\overleftarrow{b}(i) \in \{\mathsf{n}, \mathsf{i}\} \;\Rightarrow\; b(i) \in \{\mathsf{i}, \mathsf{s}\}$$

Note also that all three implications could have been equivalences without changing the operation.

### Acquire

The second operation, acquire, is executed when a read is about to start. It adds the new reader's name, passed as a parameter, to the record of active readers and reassigns status flags as necessary.

If there is a buffer currently being read, then the new read also begins to read that same buffer and no status change is required. Otherwise the new read starts on the buffer with newest data, status n, and reassigns the status of that buffer to m.

The operation can only be executed in these two situations and this information is recorded in the precondition which requires that there is either a status m or status n buffer. The precondition also records the fact that the operation is only required to function when the new reader is not already in the set of readers.

Note that in selecting which buffer is to be read, it is not always possible to choose the buffer with newest data. This situation occurs when there are currently buffers with both status m and n, which arises when the data in the n buffer has become available since the start of an ongoing read, that is, when there has been a *slave* since an *acquire* for which there has not yet been a corresponding *release*. In this situation, were the new master to begin reading the n buffer, there would then be two buffers reserved for reading. Consequently, should another *slave* now occur, attempting to preserve this new data would leave no buffer being available for another write to start, thus contradicting one of the fundamental requirements of the protocol: that processors should never have to wait to gain access to buffers. The invariant prevents this possibility by requiring that there is always at most one buffer with status $\mathsf{m}$.

$$
\begin{array}{l}
acq\ (l{:}\ MName) \\
\mathsf{ext\ wr}\ b \quad :\ Status^* \\
\quad\ \ \mathsf{wr}\ ms\ :\ MName\text{-}\mathsf{set} \\
\mathsf{pre}\ l \notin ms \wedge \\
\quad\ \ \exists i \in \{1,2,3\} \cdot b(i) = \mathsf{n} \vee b(i) = \mathsf{m} \\
\mathsf{post}\ ms = \overleftarrow{ms} \cup \{l\} \wedge \\
\qquad \forall i \in \{1,2,3\} \cdot \\
\qquad\qquad \mathsf{if}\ \overleftarrow{b}(i) = \mathsf{n} \wedge \overleftarrow{ms} = \{\,\}\ \mathsf{then}\ b(i) = \mathsf{m}\ \mathsf{else}\ b(i) = \overleftarrow{b}(i)
\end{array}
$$

It is worth observing that the last line of the postcondition could have been written as

$$\mathsf{if}\ \overleftarrow{b}(i) = \mathsf{n}\ \mathsf{then}\ b(i) \in \{\mathsf{n},\mathsf{m}\}\ \mathsf{else}\ b(i) = \overleftarrow{b}(i)$$

or simply as

$$\overleftarrow{b}(i) \neq \mathsf{n} \wedge ms \neq \{\,\} \ \Rightarrow\ b(i) = \overleftarrow{b}(i).$$

The apparent non-determinism in the alternatives is illusory because the invariant will ensure that there is no real choice as to the status to assign to any buffer that previously had status $\mathsf{n}$. However, the longer and apparently stronger postcondition is preferred as the shorter versions seem to be more cryptic.

### Release

The release operation is executed when a reading, master processor finishes its read. The name of the processor is removed from the set of readers and again, status flags are reassigned as required.

If this master is not the last one currently reading, then no change is required to the status flags. However, if this is the last master currently reading the $\mathsf{m}$ buffer, then this buffer must have its flag reassigned. There are two possibilities. On the one hand, should there be another buffer with status $\mathsf{n}$ available at this time, that is if a write has been completed since the current "chain of reads" began on this buffer, then the $\mathsf{m}$ buffer no longer contains the most recent data and so should now be set to $\mathsf{i}$. On the other hand, if there has been no write since the chain of reads began, and hence there is no $\mathsf{n}$ buffer available, the $\mathsf{m}$ buffer contains the most recent data and its status should be reset to $\mathsf{n}$.

$rel\ (l{:}\ MName)$
$\mathsf{ext\ wr}\ b\quad : Status^*$
$\quad\mathsf{wr}\ ms\ : MName\text{-}\mathsf{set}$
$\mathsf{pre}\ l \in ms$
$\mathsf{post}\ ms = \overleftarrow{ms} - \{l\} \wedge$
$\quad \forall i \in \{1, 2, 3\} \cdot$
$\qquad \mathsf{if}\ ms = \{\,\} \wedge \overleftarrow{b}(i) = \mathsf{m}$
$\qquad \mathsf{then}\ b(i) \in \{\mathsf{n}, \mathsf{i}\} \wedge count(\mathsf{n}, b) = 1$
$\qquad \mathsf{else}\ b(i) = \overleftarrow{b}(i)$

Again there is some choice as to how much of the information that is deducible from the invariant should be made explicit in the postcondition. For example, the first conjunct of the 'then' clause, $b(i) \in \{\mathsf{n}, \mathsf{i}\}$, could have been omitted as no other possibilities are permitted by the invariant, or alternatively, the whole 'then' clause could be replaced by a more explicit form

$$\mathsf{if}\ \exists j \in \{1, 2, 3\} \cdot \overleftarrow{b}(j) = \mathsf{n}\ \mathsf{then}\ b(i) = \mathsf{i}\ \mathsf{else}\ b(i) = \mathsf{n}$$

It is debatable which gives the clearer specification.

### 2.2.3 Discussion

**Invariants and postconditions**

In this presentation, the invariant has been used to convey quickly an understanding of the reachable values of the state. In VDM, the state invariant is effectively part of the state typing information and, as such, is assumed to be maintained by the operations.

It is this implicit maintenance of the invariant which leads to the choices discussed above concerning the degree to which the information that is deducible from the invariant should be repeated in a postcondition. There is often some tension between the most concise form that relies on properties of the invariant for its correctness, and a longer, but more explicit form, that includes some redundant information. This choice can be seen as an opportunity to perform some validation of the specification by proving the stronger forms from the weaker.

Which form is chosen may make a significant difference to the complexity of the proofs in reification: the form that most clearly conveys the information may not be the form that will be most usable in proofs. Indeed, the stronger form is more likely to be helpful when the specification is being proved to be a reification of another, and the weaker form when it is itself being reified. By proving one form from the other, one can move some of the burden of proof that otherwise might have arisen when justifying a data reification into the validation of the single specification. This may well be to some advantage as reasoning in the context of a single specification is likely to be less complex than reasoning about a pair of specifications.

On the other hand, considering the invariant as part of the state type can be inconvenient when it comes to developing the operation towards code. If the separate components of the state type are to be understood to represent separate program variables, then it is unreasonable to expect that the intermediate states visited during the execution of the operation will satisfy the invariant. In this example, none of the operations are likely to maintain the invariant during execution as the values of variables will be changed one at a time and therefore intermediate states will temporarily break the invariant. Consequently, for the purposes of algorithm reification, it can be convenent to consider the invariant as part of the precondition and postcondition predicates, rather than as intrinsic to the state type. This is the approach taken, for example, in Morgan's refinement calculus [Mor90]. This issue of how one is to interpret the invariant is discussed further in the next chapter and then arises again in Chapters 6 and 7.

## 2.3 An alternative model for MSMIE

The above specification is based on the state recording the status of each buffer. In effect, the state is a map from each buffer to its status. Now, since there is never more than one buffer with a given status, it is possible to invert the map and think of the state as a mapping from each status to a buffer or nil. This leads to an alternative specification which could yield a more efficient basis for an implementation. This change of variables also makes it possible to give read and write constraints more precisely and raises some interesting questions concerning their use.

### 2.3.1 The state

The state is modelled by giving which buffer currently has status $\mathsf{s}$ and which (if any) have status $\mathsf{m}$ or $\mathsf{n}$. The invariant ensures that there are never two buffers with the same status.

types

$$BName = \{1, 2, 3\}$$

$$MName = \mathsf{token}$$

$$
\begin{array}{l}
\mathsf{state}\ \Sigma_1\ \mathsf{of} \\
\qquad \begin{array}{rcl} s & : & BName \\ m & : & [BName] \\ n & : & [BName] \\ ms & : & MName\text{-}\mathsf{set} \end{array} \\
\mathsf{inv}\ mk\text{-}\Sigma_1(s, n, m, ms) \ \triangleq\ (\mathsf{m} = nil \ \Leftrightarrow\ ms = \{\,\}) \wedge \\
\qquad\qquad\qquad\qquad\qquad\quad nil\text{-}or\text{-}different([s, m, n]) \\
\mathsf{init}\ mk\text{-}\Sigma_1(s, n, m, ms) \ \triangleq\ mk\text{-}\Sigma_1(1, \mathsf{nil}, \mathsf{nil}, \{\,\}) \\
\mathsf{end}
\end{array}
$$

where[4]

$$nil\text{-}or\text{-}different : [A]^* \rightarrow \mathbb{B}$$

$$nil\text{-}or\text{-}different(l) \ \triangleq\ \forall i \in \mathsf{inds}\, l \cdot l(i) = nil \vee l(i) \notin \mathsf{rng}\,(i \mathbin{⩤} l)$$

The two models of the state are related by the following retrieve function which is one-to-one.

$$retr : \Sigma_1 \rightarrow \Sigma$$

$$
\begin{array}{l}
retr(mk\text{-}\Sigma_1(s, n, m, ms)) \ \triangleq \\
\qquad \mathsf{let}\ b : Status^* \\
\qquad \mathsf{be\ s.t.}\ \forall i \in \{1, 2, 3\} \cdot \begin{array}[t]{l} (s = i \ \Leftrightarrow\ b(i) = \mathsf{s}) \wedge \\ (n = i \ \Leftrightarrow\ b(i) = \mathsf{n}) \wedge \\ (m = i \ \Leftrightarrow\ b(i) = \mathsf{m}) \wedge \\ (i \notin \{s, n, m\} \ \Leftrightarrow\ b(i) = \mathsf{i}) \end{array} \\
\qquad \mathsf{in}\ mk\text{-}\Sigma([b(1), b(2), b(3)], ms) \\
\qquad \mathsf{end}
\end{array}
$$

### 2.3.2 The operations

#### slave

The definition of *slave* can now be made extremely concise.

[4] Again the sequence is viewed as a map from natural numbers and so map domain deletion, $⩤$, is used to remove an element from the sequence.

$$
\begin{array}{l}
slave\ () \\
\mathsf{ext\ rd}\ m\ :\ [BName] \\
\quad \mathsf{wr}\ n\ :\ [BName] \\
\quad \mathsf{wr}\ s\ :\ BName \\
\mathsf{pre\ true} \\
\mathsf{post}\ n = \overleftarrow{s}
\end{array}
$$

Two simple validation properties for *slave* are worth mentioning as they highlight the use of the frames: $m$ is unchanged as it is read-only, and $s$ is non-deterministically assigned to any value permitted by the invariant.

$$
\begin{array}{l}
m = \overleftarrow{m} \\
s \in BName - \{n, m\}
\end{array}
$$

The interaction between invariant and externals is interesting. Firstly, although inspection of the predicates does not indicate that $s$ must be written, read and write access to $s$ is required as it must be reassigned in order to maintain the invariant. Secondly, even though $m$ is not referred to in the operation predicates, read access to $m$ is required in order to be able to choose a suitable $s$ for the after state. These anomalies arise since $m$ is linked to $s$ via the invariant and the value of $s$ is not determined explicitly by the post-condition. Thus any implementation will need to read $m$ in order to ascertain what value it is valid to assign to $s$. On the other hand, in this example, read access is not required to $ms$ although $ms$ is also linked to $m$, and hence also to $s$, by the invariant. This issue is discussed further in the next chapter.

### acquire

The *acquire* operation does not require access to $s$

$$
\begin{array}{l}
acq\ (l{:}\ MName) \\
\mathsf{ext\ wr}\ ms\ :\ MName\text{-}\mathsf{set} \\
\quad \mathsf{wr}\ n, m\ :\ [BName] \\
\mathsf{pre}\ l \notin ms \wedge \neg\,(n = \mathsf{nil} \wedge m = \mathsf{nil}) \\
\mathsf{post}\ ms = \overleftarrow{ms} \cup \{l\} \wedge \\
\qquad (\overleftarrow{ms} \neq \{\,\} \ \Rightarrow\ m = \overleftarrow{m} \wedge n = \overleftarrow{n}\,) \wedge \\
\qquad (\overleftarrow{ms} = \{\,\} \ \Rightarrow\ m = \overleftarrow{n} \wedge n = \mathsf{nil})
\end{array}
$$

Interestingly, the third conjunct of this postcondition may be considered redundant. When $\overleftarrow{ms} = \{\,\}$, then $ms = \{l\}$ and so $m$ must be assigned a non $\mathsf{nil}$ value. Now, as read access to $s$ is prohibited, the only buffer that we can be sure is not already in use is that previously assigned to $n$. So any correct implementation that respected the frames of reference must assign this buffer to $m$. Then the only remaining possible value for $n$

is nil. However, to hide so much information in the externals clauses would seem to be counter-productive.

**release**

The read and write accesses of *release* are the same as those of *acquire*

$$
\begin{array}{l}
rel\ (l\colon MName) \\
\mathsf{ext}\ \mathsf{wr}\ ms \quad : MName\text{-}\mathsf{set} \\
\qquad \mathsf{wr}\ n, m\ : [BName] \\
\mathsf{pre}\ l \in ms \\
\mathsf{post}\ ms = \overleftarrow{ms} - \{l\} \wedge \\
\qquad (ms \neq \{\,\} \;\Rightarrow\; m = \overleftarrow{m} \wedge n = \overleftarrow{n}\,) \wedge \\
\qquad (ms = \{\,\} \wedge n \neq \mathsf{nil} \;\Rightarrow\; n = \overleftarrow{n} \wedge m = \mathsf{nil}) \wedge \\
\qquad (ms = \{\,\} \wedge n = \mathsf{nil} \;\Rightarrow\; n = \overleftarrow{m} \wedge m = \mathsf{nil})
\end{array}
$$

In this case the variables of the specification and those of the implementation are the same. Even though $s$ is linked by the invariant to $n$ and $m$, read access to $s$ is not needed – it is possible to prove satisfiability and validity of implementations without knowledge of the value of $s$.

### 2.3.3 Discussion

**Read frames and scoping**

In this specification, we have seen some complex interdependencies between the invariant, externals and postcondition. In *slave*, read access to one component was required although that component was not itself referred to in the specification. In *acquire*, information in the externals could be used to define a highly implicit specification. In *release* however, we saw that some components were not relevant to the operation definition even though they were related to those mentioned in the predicates.

In general, in order to ascertain the validity of possible implementations, it may be necessary to draw information from the variables in the transitive closure of the "linked by invariant" relation. Of course, this set of variables may be quite different from those that appear in the operation specification or those that will be accessed by the implementation.

Some of these issues can be clarified by considering the externals clauses not as concerning the variables mentioned in the *specification*, but rather as giving "advanced information" of what access to state variables an *implementation* of that operation can be allowed to make. This distinction separates their semantic role, giving information about access to state variables, from the syntactic role they play in binding the free variables of the precondition and postcondition.

The "linked by invariant" condition partitions the state components into independent subsets and these parts are the level of granularity at which sense can be made of the operation definition. Thus, in general, they are also the finest partition for which satisfiability and refinement can be sensibly defined.

This issue will be looked at in depth in the next chapter where more examples are given (Section 3.1.3) and where the roles of frames and invariants in algorithm refinement are considered more closely, independence is defined, and some proof rules for algorithm refinement with frames are proposed. The definition of a semantic model which supports these proof rules will be the subject of subsequent chapters.

These two specifications have shown that far from being a minor concern, there are many interesting questions of design that arise from careful consideration of the read and write frames in model-oriented operation specifications. These issues are taken up in more detail in later chapters, but for the present example we end with a general discussion of methodological questions that arise from the comparison of the model-oriented treatment of the example given here, with the previous analysis using CCS.

## 2.4 General discussion: Methodological issues

This chapter has described a model-oriented development of an application that was previously analysed through formalisation in a process algebraic notation (CCS) [BA91, BA92a, BA94]. As a comparative study it highlights the different benefits of each approach and raises some questions concerning the choice of methodogy to be used in the development process. This section gives a brief discussion of some of these concerns.

In deciding between these two approaches to system description, *inter alia*, the specifier should clearly consider the type of system that is going to be defined and, in particular, what features of that system are to be the subject of analysis. One criterion that may be considered important is the degree of concurrency envisaged for the system, or perhaps more accurately, the degree to which it is the concurrency in the system that will be the subject of the formal analysis to be undertaken. Model-oriented (MO) methods are typically advocated for sequential systems and process-based methods for the study of concurrency.

In process algebra (PA), we know from the unfolding lemma, that it is always possible to abstract away from parallelism, that is, to view the system behaviour from the outside. We may then study whole system behaviours such as equivalences between different combinations of operations. MO methods are not so atuned to studying these behaviours, rather they are typically used to study the system at the operation and data level. Their focus is the effects of individual operations on the state or alternative possible models for the data.

There is, of course, a direct correspondence between the two views: both define an abstract machine: a state space and the possible transitions between states. The difference is merely one of presentation: MO defines, for each operation, what changes of state are possible; whereas, PA gives, for each state, what operations are possible.

The different formalisms are best suited to the different concerns that arise at different stages of the development process. Some validation of a formal description is likely to be in terms of the behaviour of the whole system: for example we may wish to assert that we cannot reach certain states, or that we cannot reach a state when no operation can apply. Other validation statements are about individual operations, for example, if some condition holds before an operation then it will also hold afterwards. In PA we have specific calculi, such as Hennessey-Milner logic [HM85] and its modal extension [BA92b] for validating whole system properties. The equivalent methodology does not seem to have been worked out for MO methods.

On the other hand, issues surounding data refinement have been largely ignored in PA as the reduction of the value passing calculus to the pure calculus side-steps such issues. It would seem to be an obvious step forward to develop a formalism that combines the benefits of the two camps.

It is quite simple to imagine a formalism that is a hybrid of the two. One could consider this to be PA with value passing or MO with interleaving. It would describe the overall system behaviour in a style like PA but without parallelism and enable “holistic” validation on this model. Then it would be possible to proceed with MO style data and algorithm refinement. During the design process, possible parallelism may re-emerge. Data refinement can lead to the identification of independent parts of the machine that may be implemented separately, algorithm development may lead to code where temporal ordering of statements is unimportant. In order to allow such underspecification in algorithm refinement, “framing” the area of influence of each operation is vital in order to demonstrate properties of non-interference.

The duality between model-oriented and process-algebraic specifications will arise again in Chapter 5 when an interesting correspondence between read frames and bisimulation is observed. For the present, we simply remark that read and write frames are clearly a means by which non-interference of operations can be specified.

## 2.5 Summary

This chapter has raised many issues concerning the model-oriented specifiation of a software module. Clearly, not all these issues can be pursued in a single work. This section summarises the points raised and indicates those which will be the focus of further attention in the course of this thesis.

**The role of invariants.** How should one interpret the invariant?

- The invariant can be used to convey quickly a level of understanding of the specification by indicating which states are in some sense "reachable".
- On the other hand, it is unreasonable to expect intermediate states during operations to satisfy the invariant.

**Postconditions and the invariant.**

- Postconditions which implicitly maintain the invariant provide the opportunity to validate an operation specification by proving a more explicit form of a postcondition from a more implicit one.
- Both the weaker and stronger forms can subsequently be useful in the verification of reifications.

**Externals.** There is a subtle interdependcy between the externals, postconditions and invariant.

- The meaning of an operation specification, that is the validity of putative implementations, can depend on the values of variables not mentioned in the predicates.
- The variables appearing in the pre/post predicates may not necessarily be those that appear in the implementing code.
- Where there is such a difference, externals clauses can be seen as indications of the variables that should appear in code rather than those that appear in the specification.

**Model-oriented versus process-oriented approaches.** As specifications of the external behaviour of a system, there is a duality between the MO and PA styles of presentation of state machines. No calculus for the validation of "global properties" has thus far been developed for model-oriented specifications.

Though all of these themes are worthy of further scrutiny, this thesis will primarily focus on the first three, the use of invariants, frames and postconditions.

# Chapter 3

# A Proof Theory for Algorithm Refinement with Read and Write Frames

*"A mathematical proof should resemble a simple and clear-cut constellation, not a scattered cluster in the Milky Way."*

## 3.1 Introduction

The read and write frames of reference variables used in the VDM style of operation decomposition serve two purposes. Syntactically, they bind the variables that occur in the predicates of the operation specification; and semantically, they record what access to the state an implementation can be allowed to make. This chapter examines the use of frames in these two roles, in particular considering the case when there is an invariant on the state. It argues that the two roles can be usefully distinguished.

A model for operation specification and refinement is developed where full account is taken of both syntactic and semantic roles of the frames. Consideration of their syntactic role gives rise to the idea that a part of the state can be independent of the rest with respect to the invariant that is in force: a definition of independence is given together with a syntactic sufficient-condition for it. The semantic role of the frames is examined in connection with notions of satisfiability and satisfaction. Satisfiability is modified to capture the restricted possible interdependencies between read and write variables and the rules for satisfaction are given that restrict the usual relation in order to respect the constraints on implementations imposed by the frames.

This chapter outlines a proof theory for algorithm refinement with read and write frames. Subsequent chapters provide a semantic framework in which the proof rules presented here can be justified.

### 3.1.1 Motivation

As discussed in Chapter 1, when developing a formalism of software specification and refinement, the issue of compositionality must be central.

Following the process of top-down design, at any stage we have a specification of a system component and the design task is to develop a component that satisfies that specification. Typically the design will be the composition of several smaller components that, at this stage, will only be specified and which will later be developed independently themselves.

The problem can be described as follows. If we require that a specification $S$ is to be refined by a design $D$, written

$$S \sqsubseteq D$$

then we can build a high level design which states that $D$ should be some composition, $C$, of components $S_1 \ldots S_n$ which are themselves only specified at this stage,

$$D = C[S_1, \ldots, S_n]$$

such that, if each component is itself refined by a design,

$$S_i \sqsubseteq D_i$$

then the original specification must be satisfied by the composition of those designs.

$$S \sqsubseteq C[D_1, \ldots, D_n]$$

Compositionality states that the development of each $S_i$ should be able to proceed independently of the others.

Here we consider two issues that arise when examining compositionality in the process of algorithm refinement, taking as a starting point the VDM style of operation decomposition given, for example, in [Jon90b]. The first consideration is that each sub-development, $S_i \sqsubseteq D_i$, should be truly independent of the rest. That is, that sufficient information should be contained in each $S_i$ to characterise its valid implementations. In particular, attention is paid to the case where there is an invariant relating possible values of components of the state. The second issue is the matter of state access: the so called *frame problem* addressed by the "externals" clauses in the operation definition. Typically, each of the smaller components, $S_i$, involves just part of the state available to $S$, so that in the development of that $S_i$ we want only to concern ourselves with that part of the state and should not have to look outside for contextual information. In VDM, the frame of reference of an operation is given explicitly by the read and write access conditions.

Of course the two issues are related since what is a sensible frame for an operation depends crucially on any invariant that may be in force on the state. The interaction of frames and invariants motivates us to distinguish the syntactic and semantic roles played by the read and write frames. The separation of these two roles yields a more expressive notation where read and write frames are freed from their use as syntactic binders for the variables in the specification and can instead play their intended role as indications of the constraints on the accesses that should be made by an implementation. The ramifications of this interpretation are examined and lead to a modified notion of satisfiability, that captures the restricted possible interdependencies between read and write variables, and a restricted satisfaction relation that respects the constraints on implementations imposed by the frames. This extra expressive power comes at the expense of some prolixity – a third frame is introduced into the definition of an operation – however, as the model proposed is envisaged as the basis for machine based support, it is anticipated that the extra complexity could be handled automatically and only come to the fore when explicitly relevant.

The rest of this section outlines the background to the work presented and gives two motivating examples which expose some ambiguities that can arise from the interaction of the read and write frames with the invariant. Section 2 develops a model for state-based operations with read and write frames and invariants. It gives a definition of when a part of the state is independent with respect to the invariant which is then used in an examination of the role that the read and write frames play in satisfiability. The third section presents some proof rules for operation decomposition where the usual satisfaction relation is restricted in order to respect the constraints on implementations imposed by read and write frames. Clearly, these frames are of little relevance in the presence of data refinement, so the treatment here only considers algorithm refinement; data refinement and operation modelling are considered in [BR91] and [BFL$^+$94]. The last section briefly discusses the issues raised in a broader perspective.

### 3.1.2 Background

**Operation decomposition in VDM.**

The starting point for the work given here is the treatment of operation decomposition in VDM given in [Jon87, Jon90b]. In these works a clear exposition of the underlying principles is given through examples and a set of proof rules for verification of refinements. These rules are proved correct with respect to a denotational semantics. We take on board the arguments given there advocating the convenience of having postconditions relating before and after states and do not reiterate those arguments here. The rules do not, however, explicitly handle the frame of reference variables. An extension of the model is given in [Mil88], however, the rules there are rather cumbersome and no attempt to justify them is made.

[AK89] is perhaps the most detailed treatment of operation decomposition in the VDM style. A denotational semantics is given for a language that extends that of [Jon90b] with arrays and procedures. A proof theory for this language is given and is shown to be sound and relatively complete with respect to the denotational semantics. In order to cater for blocks and procedures with static scoping, the usual Hoare triple is augmented by a syntactic environment with static and dynamic components. The static part of the environment may be compared with our set of read variables, and the work does indeed consider extending and contracting this frame. However, it does not distinguish the read-only from the read-write variables.

**Other approaches to operation decomposition.**

There is a large body of work stemming from [Dij76] that considers verification of programs in the framework of a weakest precondition semantics. Mostly, this work considers postconditions of one state which leads to some elegant mathematics.

Some more recent works add specifications to the language bringing them closer to present concerns. [Mor87] gives an elegant treatment with "prescriptions" which makes use of higher ordinals to handle unbounded non-determinism. [Mor88b] includes a similar "specification statement" and explicitly handles the write variables and also postconditions of two states. Many previous works are brought together in [MRG88] and the central ideas are presented in textbook form in [Mor90] which gives a formalism for algorithm refinement via a set of proof rules defining valid refinements. In this treatment, all state variables are global with respect to read access, and a single "frame" is given that denotes those variables that are able to be written.

[Bac88] and [BvW90] do consider the frame of free variables available for use in expressions and also cater for specification statements through non-deterministic assignment statements. However, this work does not distinguish the read-only from the read-write variables. There, the frame cannot contract during development. This has the effect that $x := x$ is not refined by skip, since skip has a smaller read frame. Thus this frame is somewhat different from the read frame considered here.

In the "B" method [Abr95, ALNS91], a great deal of attention is paid to issues of framing and to the incremental construction of specifications and their refinements through the use of structured "abstract machines". Several forms of hierarchical composition of machines are provided embodying both full-hiding, where variables of the included machine are accessible only through the operations of that machine, and semi-hiding, where state variables can be read directly but written only through the operations. Thus, the frames for possible read and write access of operations are given through the structuring of a machine, whereas the actual read and write accesses made are given implicitly through the definition of operations as generalised substitutions.

### 3.1.3 Examples

Two example operations are given that motivate the usefulness of a third frame. They both act on the same state, which although very simple, is already complex enough to expose the issues with which we wish to deal:

$$\begin{aligned} \mathit{State} :: \ & a : \mathbb{N} \\ & b : \mathbb{N} \\ & c : \mathbb{N} \end{aligned}$$

$$\begin{aligned} & \mathit{inv\text{-}State}(a, b, c) \ \triangleq \\ & \quad a \in \{0,1\} \wedge \\ & \quad b \leq a \wedge \\ & \quad c \in \{1,2\} \end{aligned}$$

A quick syntactic inspection of the clauses in the invariant shows us that the components $a$ and $b$ are in some way "linked" whereas $c$ is independent of $a$ and $b$.

**Example 1, choose_b.**

Let us consider the specification of an operation[1] that writes $b$, say $\mathit{choose\_b}$.

$\mathit{choose\_b}$

ext rd $b$ : $\mathbb{N}$

    wr $b$ : $\mathbb{N}$

post true

Clearly, because of the invariant, the operation is not free to choose any $b$: $\mathbb{N}$, rather it needs to ensure that the invariant is maintained, i.e. $b \leq a$. So one might be led to believe that such an operation specification should be meaningful only if it also has read access to $a$. On the other hand, it is possible to construct a correct implementation of the operation which does not require read access to $a$, for example both $b := 0$ and $b := b$ ensure that the invariant is maintained. Which of the possible implementations are valid depends on how we interpret the read frame when an invariant is in force.

Let us consider the first of these alternatives: we will restrict what we consider to be a well-formed operation so that state access is always "sensible" with respect to the invariant.

As a first cut, let us say that an operation specification must have read access to (at least) all the fields which appear in the same conjunct of the invariant that a write field appears in. Thus the operation reads sufficient of the state to ensure that all the relevant clauses in the invariant are maintained. (There is no need to consider those clauses not mentioning any writes since they will not be affected by the operation.)

This resolves the problems with the example $\mathit{choose\_b}$ above which becomes:

[1]Unlike the standard usage, we choose to record variables which are read-write in both rd and wr clauses.

$choose\_b'$
ext rd $a, b$ : $\mathbb{N}$
    wr $b$ : $\mathbb{N}$
post true

This is interpreted as if the relevant clauses, $b \leq a$ and $a \in \{0, 1\}$, were added to the precondition and postcondition so, for example, the satisfiability criterion would then quantify over the read frame and include the appropriate clauses of the invariant:

$$\forall a: N, \overleftarrow{b}: N \cdot a \in \{0,1\} \wedge \overleftarrow{b} \leq a \;\Rightarrow\; \exists b: N \cdot a \in \{0,1\} \wedge b \leq a \wedge \mathsf{true}$$

However, by restricting well-formedness in this way, we have lost some of the expressive power in the language. Perhaps we really did intend to specify that the operation should maintain the invariant while not reading $a$. That is, we did mean that the only valid implementations should be variants on $b := 0$ and $b := b$.

### Example 2, set_c.

Now consider the operation specification:

$set\_c$
ext rd $b$
    wr $c$
post $c \in \{b, b+1\}$

This is a well-formed operation by the above criterion and does indeed specify a bona-fide operation with two obvious implementations $c := b$ and $c := b + 1$. But in order to know that it is satisfiable (ie. that $c \in \{1, 2\}$) we need the information relating to $a$ that is given in the first clause of the invariant. Furthermore, apart from the two obvious implementations, if we consider the extra information about $a$ given in the first clause of the invariant, then $c := 1$ is also a valid implementation.

Thus the "available information" for determining valid implementations must include all the clauses that could give us any information about fields that appear in clauses with fields that could be written. That is, we must take the transitive closure of the "appears in a clause with" relation. This closure partitions the state into parts which we will later call *independent* and in order to characterise the possible implementations of the specified operation we may need all the information from the invariant concerning fields that are "connected" to any field in the frame. So for an operation specification to have a well-defined meaning it must be considered in the context of all the parts of the state connected to variables appearing in its definition.

However, there is still good reason to keep the smaller read frame since it can give information as to what implementations are intended to be valid. In this case, for example, although we require access to $a$ in order to ascertain what the valid implementations are, it is not intended that $c := a$ should itself be a valid implementation, as would be the case if the read frame were extended to include $a$. Thus expanding the read frame to include all connected fields would change the meaning of the operation specification.

The approach investigated here will be to give a third frame for each operation. This frame plays the "syntactic" role of carrying the declarations of all fields related to the fields in the read and write frames. The read and write frames are thus freed to play their "semantic" role: to give information about the access permitted of valid implementations.

## 3.2 A model of operations with frames

### 3.2.1 Operation definitions

We have motivated the need for a third frame which will simply be called *the frame*, as opposed to *the reads* and *the writes*. This frame must be an *independent part* of the state in a sense which is defined below. In order to make the operation definition independent of the state, it will also carry those clauses of the invariant that pertain to it.

Thus an operation definition is composed of six parts: the *frame* which carries the declaration of all the variables that are in scope and binds all the free variables that appear in the rest of the operation definition, the *invariant* which contains contextual information about these variables, the *reads* and *writes* which give information about access to the reference variables that must be maintained by any implementation and the *pre* and *post* which have their usual VDM interpretation, that is as if the invariant were conjoined to them[2].

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

where

$$
\begin{array}{l}
inv\text{-}OpDef(mk\text{-}OpDef(F, I, R, W, P, Q)) \quad \triangleq \\
\qquad R \subseteq dom(F) \wedge \\
\qquad W \subseteq dom(F) \wedge \\
\qquad I \colon Exp(dom(F)) \wedge \\
\qquad P \colon Exp(dom(F)) \wedge \\
\qquad Q \colon Exp(dom(F) \cup \overleftarrow{dom(F)})
\end{array}
$$

[2]The exposition here does not deal with operations with parameters and results.

The notation in the last three clauses of the invariant is explained in the next section.

Note that we do not insist that the free variables of $P$ and $Q$ are restricted to $R$ and $W$ in the usual way, nor do we require that $R \supseteq W$. Although the usefulness of write-only variables in operations is debatable, we will see in Chapter 5 how their consideration is helpful in order to give an orthogonal treatment to the read and write frames.

In what follows, we will often use $F$ in place of $\mathsf{dom}\, F$ for the set of identifiers in the overall frame. The meaning will always be clear in context.

It is also a requirement that the overall frame of an operation definition be an "independent part" of the frame of the overall state. This requirement will later form part of the definition of the satisfaction relation but before we consider it in more detail we introduce some notation.

### 3.2.2 Notation

**Hooking.**

If $S$ is any set of identifiers, say

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

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

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

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

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

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

$$E : Exp(S)$$

and $S_1 \subseteq S$, then

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

Thus:

$$\overleftharpoon{E}^{S_1} : Exp(\overleftharpoon{S}^{S_1})$$

**Identity.**

We define an informal meta-notational shorthand for saying that the variables in a set are unchanged. $Id_S$ is simply the conjunction of clauses each stating that a variable in $S$ is unchanged.

$$Id_S \triangleq \bigwedge_{x_i \in S} x_i = \overleftarrow{x_i}$$

**Quantification.**

Let $F$ be a composite type

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

and let $S$ be a subset of the fields of $F$

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

Let $E$ be an expression with free variables in $S$

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

then we use the notations

$$\forall \mathsf{S} \cdot E \ , \ \exists \mathsf{S} \cdot E$$

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

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

Similarly

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

is a shorthand for the corresponding hooked formula.

**Invariants.**

Since we will be analysing the role of invariants on composite types it will be convenient to write them explicitly, $F \dagger I$, when we mean the type restricted by the invariant and we leave the undecorated type name, $F$, to denote the "free" type. Thus, for example:

$$\forall \mathsf{F} \dagger \mathsf{I} \cdot E \ \triangleq \ \forall \mathsf{F} \cdot I \ \Rightarrow \ E$$

### 3.2.3 Independence of a part of the state

We will first give a "semantic" definition of when a part of the state is independent with respect to the invariant, before reverting to a syntactic condition.

**Definition.**

Let $S \subseteq F$ be a subset of the fields of a composite type with invariant, $F \dagger I$, and let T be the rest of the fields, $T = F - S$.

The part $S$ of $F$ is said to be an *independent part*, written $S \overset{ind}{\subseteq} F$, if and only if, swapping the $S$ parts of two states that each satisfy the invariant maintains the invariant. That is if:

$$\forall \mathsf{F}, \overleftarrow{\mathsf{F}} \cdot I \wedge \overleftarrow{I} \;\; \Leftrightarrow \;\; \overleftarrow{I}^{S} \wedge \overleftarrow{I}^{T}$$

This can be shown to follow from the following, more succinct, definition which does not require one to show $\overleftarrow{I}^{T}$:

$$S \overset{ind}{\subseteq} F \;\; \triangleq \;\; \forall \mathsf{F}\dagger\mathsf{I}, \overleftarrow{\mathsf{F}\dagger\mathsf{I}} \cdot \;\; \overleftarrow{I}^{S}$$

The following theorem states that a part is independent exactly when it is possible to write the invariant in such a way that the independence of that part is syntactically obvious.

**Theorem.**

A part $S$ of composite type $F \dagger I$ is an independent part of $F$ if and only if there are predicates $I_S\colon Exp(S)$ and $I_T\colon Exp(F - S)$ such that

$$F \dagger I = F \dagger (I_S \wedge I_T)$$

**Proof.**

The proof proceeds by construction of the two required predicates. The key definition, the *restriction* of the invariant to a part of the state, is given below. The proof then follows easily by showing that the required predicates are the obvious restrictions of the invariant. Details are given in Appendix A.1.1.

**Definition.**

For a part $S$ of composite type $F \dagger I$, define $I_S$, the invariant *restricted to* $S$ by

$$I_S \triangleq \exists \mathsf{F}-\mathsf{S} \cdot I$$

With this definition the following corollary gives a criterion for independence.

**Corollary.**

$$S \overset{ind}{\subseteq} F \quad \textit{iff} \quad \forall \mathsf{F} \cdot I_S \wedge I_{F-S} \Leftrightarrow I$$

It is easy to show that the syntactic condition for independence described in example 2, that is, the closure of the relation "appears in the same clause as", is a sufficient condition for independence. In practical cases, it is likely that this syntactic condition will yield a sufficiently fine partition of the state for refinement. Naturally, it would be simple for a support tool for specification to use this condition to infer a default frame and local invariant for an operation from the reads and writes and the state invariant.

This definition of independence is, of course, simply a statement that the state space should be the cartesian product of the spaces spanned by the independent part and by the rest. As such, independence may not be respected by a change of variables. That is, an independent part of the state need not be independent should a different, but equivalent, data model be chosen. As stated earlier, it is only sensible to "frame" an operation over an independent part of the state and thus greater independence between state components may be a sign of a cleaner data model. Within the frame, the reads and writes give extra constraints on what constitutes a valid implementation and thus restrict the usual definition of the satisfaction relation.

### 3.2.4 Satisfiability

Given an operation specification with a frame that is an independent part of the overall state, and given that the restricted invariant is also given as part of the operation specification, it is possible to consider satisfiability (and refinement) using only local information.

The standard satisfiability condition, roughly stated, says that for any initial state satisfying the precondition, there must be a final state that satisfies the postcondition. For state $F$ with invariant $I$ and an operation with precondition $P$ and postcondition $Q$, satisfiability can be stated formally as follows:

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

However, this tells us nothing about the fact that the implementation must respect the access conditions given by the read and write frames. How should the condition be generalised to accommodate this information?

A first attempt to rectify this might be simply to quantify over the read and write frames only, with a statement such as "for all values of the reads there must exist possible values of the writes such that ...". Perhaps, for the above operation with the reads $R$ and writes $W$, we might consider[3]:

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

However this is also inadequate since the invariant might mention variables outside the read and write frames which would then be free in the formula.

Neither would it be sufficient simply to expand the "frame" of universal quantification to encompass a wider set of variables: this would lose the intention that the choice of writes should be made without recourse to the value of variables outside the read frame.

The correct formulation is a condition that is "scoped over" the whole frame of the operation, but in which the quantifiers for the parts inside and outside the reads and writes have been interspersed to reflect the fact that values assigned to the writes can only depend on the part inside the reads and that these values should be valid irrespective of the values of fields outside the reads. This idea can be captured formally by rearranging the order of the quantifiers in the formula:

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

The position in this formula of the existential quantification over $W$ captures the fact that the values given to the writes can depend only on the reads and must be valid whatever values the unread variables may have.

## 3.3 Algorithm refinement

In order to generalise the definition of refinement to respect the access constraints given by the reads and writes, it is necessary to restrict the usual satisfaction relation. Rather than attempt to give a comprehensive proof theory for refinement here, attention of focused on those rules which relate to the read and write frames. For notational brevity, in this section we will write $(F, I, R, W, P, Q)$ for $mk\text{-}OpDef(F, I, R, W, P, Q)$.

[3] For variables outside the writes, where we know hooked and unhooked values are equal, we have the freedom to use hooked or unhooked variables as we like. Unlike the standard usage, in this section we choose to use the hooked names, so hooked variables appear for all the reads whereas unhooked variables only appear for the writes.

**Weaken Pre and Strengthen Post.**

The rules for weakening the precondition and strengthening the postcondition contain no surprises. They are the obvious extensions of the usual rules [Jon87].

$$\frac{P_A \wedge I \;\Rightarrow\; P_C}{(F, I, R, W, P_A, Q) \sqsubseteq (F, I, R, W, P_C, Q)}$$

$$\frac{\overleftarrow{P} \wedge \overleftarrow{I} \wedge Q_C \wedge I \wedge Id_{F-W} \;\Rightarrow\; Q_A}{(F, I, R, W, P, Q_A) \sqsubseteq (F, I, R, W, P, Q_C)}$$

**Sequential Composition.**

To justify a sequential compositition of statements, we first manipulate the read and write frames so that they are the same for the two component operations. Thus the proof rule for sequential composition assumes that all the operations have the same frames.

$$\frac{\begin{array}{c}\overleftarrow{P \wedge I} \wedge Q_1 \wedge I \;\Rightarrow\; P_2 \\ Q_1 ; Q_2 \;\Rightarrow\; Q\end{array}}{(F, I, R, W, P, Q) \sqsubseteq (F, I, R, W, P, Q_1);(F, I, R, W, P_2, Q_2)}$$

**Conditional Composition.**

A similar approach is taken for conditional composition, however in this case the rule must also ensure that the test does not refer to any variables outside the read frame[4].

$$\frac{B \colon Exp(R)}{(F, I, R, W, P, Q) \sqsubseteq \begin{array}{l}\textsf{if } B \\ \textsf{then } (F, I, R, W, P \wedge B, Q) \\ \textsf{else } (F, I, R, W, P \wedge \neg B, Q)\end{array}}$$

**Contract Reads and Writes.**

An implementation that achieves a specification whilst reading or writing fewer variables than the specification permits is obviously correct. Because the overall frame is unchanged, we can shrink the reads and writes without worrying about variables becoming unquantified, however we must ensure that the read frame still contains the write.

$$\frac{R_A \supseteq R_C \supseteq W}{(F, I, R_A, W, P, Q) \sqsubseteq (F, I, R_C, W, P, Q)}$$

$$\frac{W_A \supseteq W_C}{(F, I, R, W_A, P, Q) \sqsubseteq (F, I, R, W_C, P, Q)}$$

[4]If we are concerned with issues arising from the use of a three valued logic, we may also wish to require that $P \;\Rightarrow\; (B \vee \neg B)$. However such matters are not relevant to the discussion here.

**Contract Frame.**

We can shrink the overall frame provided that the new frame is an independent part of the old, that no variable mentioned falls out of scope, and that the invariant is restricted to the new frame.

$$\frac{F_A \overset{ind}{\supseteq} F_C,\ (F_C, I_{F_C}, R, W, P, Q)\colon OpDef}{(F_A, I, R, W, P, Q) \sqsubseteq (F_C, I_{F_C}, R, W, P, Q)}$$

Note that the second condition described above is captured in the second hypothesis by insisting that the new *OpDef* is well-formed, whilst the third is ensured by the fact that $I_{F_C}$ in the conclusion is the invariant restricted to the concrete frame $F_C$.

**Expand Frame.**

We can expand the overall frame, that is, declare new local variables. The new variables will not appear in the abstract specification though they can later appear in the implementation. Naturally, since they are to be used as local variables, they are available for reading and writing.

$$\frac{\mathsf{dom}\, F \cap \mathsf{dom}\, S = \{\,\}}{(F, I, R, W, P, Q) \sqsubseteq \mathsf{var\ S\ in}\ (F \cup S, I, R \cup S, W \cup S, P, Q)\ \mathsf{end}}$$

The notation var S is similar to that given for quantifiers. var S stands for a sequence of declarations var $v_1\colon S_1, \ldots, v_k\colon S_k$.

**Weakening the Invariant.**

It was seen in Chapter 2, that when introducing a model of the state, a strong invariant can be a useful way to convey quickly an understanding of the 'reachable' states. The reachable states are those that can result after some sequence of operations.

However, does the invariant need to be maintained *during* the execution of each operation? If this were required, then a strong invariant could become a hinderence during algorithm refinement for it is quite possible that it would be convenient to 'visit' some states during the execution of an operation that are prohibited by the invariant. In the MSMIE example, it is certainly the case that during the re-assignment of the flags, when some have been updated and others have not, the invariant will not hold.

In Jones' treatment, the invariant is considered to be part of the state type and, since sequential composition is defined in terms of the existence of a middle state (which therefore satisfies $I$), the invariant must hold throughout execution. Thus for operation decomposition in this setting, a weak invariant is likely to be appropriate.

In Morgan's refinement calculus, a different approach is taken. The state space is the product of the spaces spanned by the individual variables and the use of an invariant is equivalent to its addition as an extra conjunct in the precondition and postcondition. However, this syntactic view of the invariant means that there is no convenient way to assert that certain states can *never* be visited.

So we see two possible roles for the invariant. The first interpretation, useful in data refinement as a way of defining application-oriented data types, we will call a *persistent* invariant. The second, more useful in operation decomposition since code manipulates the state as a set of separate variables, we will call a *transient* invariant.

A transient invariant can, however, be considered to be simply a shorthand for extra clauses in the precondition and postcondition and is thus a syntactic device which can be adopted or not with little consequence. But is there a use for a persistent invariant in operation decomposition? Indeed, it could well be that the developer intends that some states should never be visited even during the execution of the operation. (An example might be that an overflow flag should never be set.) In such cases it will be informative to be able to declare this intention by using a persistent invariant. For this reason we do consider the possibility of a persistent invariant here, assuming that any transient invariant is "expanded out" at a purely syntactic level.

The persistency of the invariant is formalised by the omission of a rule which allows the weaking of the invariant provided that precondition and postcondition are strengthened appropriately. This omission has serious repurcussions on the semantic model which are discussed further in Section 7.3.

Note that in this scenario, it is likely that prior to operation decomposition, a particular form of data refinement will be required which, although it does not change the definition of the variables, simply weakens the invariant to introduce the 'just-visiting' states. Note that this refinement will have states in the concrete model which have no counterpart in the abstract, but the retrieve function will nevertheless have to relate them in some way to states in the abstract model. In this case it might be useful to employ some device to facilitate this such as a special unreachable state in the abstract model.

**Expanding Writes.**

A similar choice exists for the interpretation of the write frame. Should the write frame be treated as a shorthand for some "rest unchanged" clauses in the postcondition or should it be used to indicate that certain variables *never* change?

Jones and Morgan both take the former view allowing the write frame to be weakened provided the relevant $x = \overleftarrow{x}$ clauses are added to the postcondition. However, in the present setting, where read and write frames are seeen as 'advanced information' about the state accesses that will be made by the implementing code, it is not appropriate to allow the write frame to be weakened as this would introduce non-write-respecting implementations. This is also the approach taken in RAISE [RLG92], although there, syntactic checks for non-writing are augmented by dynamic ones because of need to evaluate indices into arrays before disjointness of variables can be decided.

Again this decision is embodied by the lack of a proof rule. This time we do not have a rule allowing the expansion of the write frame provided the 'extended' postcondition is ensured. Again this has ramifications in the semantics which are discussed in Section 7.3[5].

**Assignments.**

It is in the rule for assignment that the read and write frames are ultimately employed: assignments can only be made to the write variables and the expression evaluated can only refer to read variables. Assuming it is well-typed[6], the assignment, $x := e$, precisely satisfies the postcondition,

$$x = \overleftarrow{e}^{x} \wedge Id_{F-\{x\}}$$

thus we get the refinement rule

$$\frac{x \in W,\ e\colon Exp(R),\ Id_{F-\{x\}} \wedge I \wedge P \ \Rightarrow\ (Q \wedge I)\left[e/x, x/\overleftarrow{x}\right]}{(F, I, R, W, P, Q) \sqsubseteq x := e}$$

## 3.4 Discussion

The motivation for the present work came from an intention to develop interactive tool support for operation decomposition in the VDM style [BR89b, BR89a]. There is much to be gained from mechanical support for this process. Indeed when developing a formalism, as well as considerations of soundness and completeness, there would seem to be good reason to keep the possibilities arising from mechanical support specifically in mind, since potential difficulties for pencil and paper methods may become unimportant when such support is provided.

As discussed earlier, compositionality is of central importance to a design methodology and the extra complexity introduced by explicitly carrying around the necessary contextual information is a small price to pay in the definition of an *abstract* model of the development process. How such information is stored and exhibited in practice is a separate concern. Indeed, to show all the context at each stage would introduce unnecessary clutter to the exposition and detract attention from the pertinent issues. Thus, it is important to know at each point in the development precisely which parts of the context have any bearing on the sub-specification under development and to be able to collect these together, if required, so that the development of that sub-specification can be considered independently.

[5]The same choice also arises in the interpretation of the read frame. Should the read frame merely constrain the values in final state so they do not depend on the variables outside the read frame or should it also insist that unread variables have no effect on the route taken to reach it?

[6]We also assume the definedness of $e$.

As pointed out earlier, the interspersing of the universal and existential quantifiers over read and write frames in the satisfiability obligation reflects the fact that the read and write frames play more than a purely syntactic role in the decomposition process. The separation of the syntactic and semantic roles of the two traditional frames can lead to some "interesting" specifications. Why should a specification be constrained to deal with the same variables as the implementation? Although there is a danger of introducing some "surprise" refinements in this way, the specifier is always at liberty to coalesce the two frames if so desired. On the other hand, judicious use of read and write frames can add considerable expressiveness to a specification. In the following chapters for example, possibilities in respect of shared state concurrency, security, and control over the removal of non-determinism will be considered.

The overall frame "scopes" the semantics of the operation specification and gives rise to a degree of freedom in the interpretation of the operation outside this frame. [Lam91] points out that an important decision in the design of his Temporal Logic of Actions, was that the semantics should be that "all can change" unless otherwise stated, as this permits more elegant laws concerning the concurrency combinators (p.54). For the sake of modularity, for instance, it may be useful to have a frame outside of which we assume nothing. In [AL90] he also says

> "A practical language must allow one to write a formula $F$ so that $y$ is a free variable of $F$ even though it does not appear in the text." (p.46)

In the context of this thesis, it is a similar idea that has motivated the separation of the 'scoping' role of the syntactic frame from its 'binding' role for the reads and writes.

# Chapter 4

# Interpreting the Read and Write Frames

*"Some measure of generality must be present in any high-class theorem but too much tends inevitably to insipidity."*

## 4.1 Introduction

We have seen how the "read" and "write" clauses in VDM implicit operation definitions 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, existing semantic models for VDM give no formal interpretation to the read frame. 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.

This chapter investigates an extension to the denotational model of operations which captures the 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 the new formalisation of the satisfiability obligation introduced in the last chapter 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 requires a more substantial change to the semantic model. This chapter introduces the basic ideas by which semantics will be given to the read frame. The next chapter presents this semantics in a more formal setting.

### 4.1.1 Background

Existing formalisations of operation decomposition in model-oriented specification pay little attention to the read frame. Jones' [Jon87] formalisation of operation decomposition in VDM incorporates 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 [ISO93].

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 $\Delta$ or $\Xi$ 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 [MRG88, 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 paid to the read frame.

In B's "Abstract Machines" [Abr95, 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.

Chapter 3 of this work introduced 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 was given.

This chapter explores a means to develop a denotational semantics for operations where an interpretation is given to both read and write frames as constraints on the interpretation of operations. This will provide a semantic model which forms the foundation of a rule-based definition of algorithm refinement given in the last chapter.

The rest of this section recaps some background material and gives the key definitions from [Jon87] and Chapter 3. Section 4.2 gives some simple examples which motivate the approach to be taken in defining the semantics. Section 4.3 gives a presentation of the main ideas of the semantics and uses it to justify the new definition of satisfiability given in Chapter 3 and, in Section 4.4, a property of non-interference. Finally, Section 4.5 discusses some shortcomings of this formalisation and indicates the direction which will be pursued in later chapters.

### 4.1.2 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).

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

For any statement $S_i$, let

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

then the domain of $M_i$ can be thought of as the set of states over which termination is possible and it is required that for all statements

$$T_i \subseteq \mathsf{dom}\, M_i$$

Refinement is defined in the usual way as a reduction in either non-determinism or undefinedness. For statements $S_1$ and $S_2$, 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 (see for example [HHJ$^+$85]).

[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 $S \lhd R \quad \triangleq \quad \{(s_1, s_2) \in R \mid s_1 \in S\}$

For the purposes of this chapter, the question of termination is not the focus of interest and thus $\mathcal{T}$ is not considered. Rather, for the present, it is assumed that, for all statements

$$T_i = \mathsf{dom}\ M_i$$

and does not change during refinement.

### 4.1.3 Operations with read and write frames

Chapter 3 argued that the syntactic and semantic roles of read and write frames can be usefully distinguished and an extra syntactic frame was proposed which binds the free variables that appear in the precondition and postcondition, hence liberating the read and write frames to play their semantic role.

Section 3.2.1 defined operations composed of six components

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

where

$$\mathit{inv\text{-}OpDef}(\mathit{mk\text{-}OpDef}(F, I, R, W, P, Q)) \quad \triangleq \quad \ldots$$

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 precondition and postcondition as usual. 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.

Section 3.2.4 gave a modified definition of satisfiability which takes account of the read and write frames 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 $\mathit{mk\text{-}OpDef}(F, I, R, W, P, Q)$ we have[2]

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

[2]The notation was explained in Section 3.2.2 and is recapped in Section 4.3.1.

## 4.2 Examples

We now present two simple examples similar to those in the last chapter 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

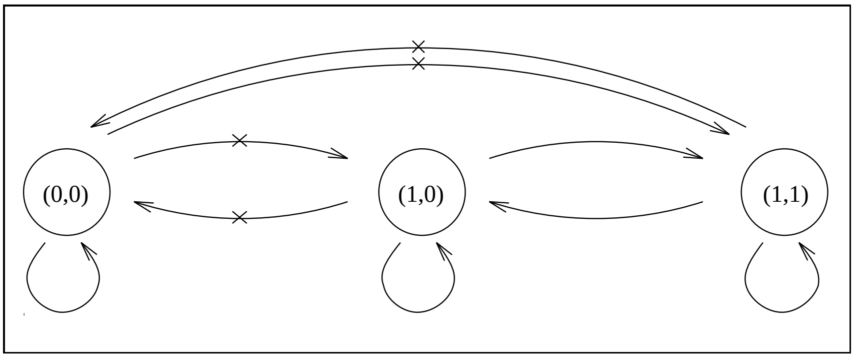
$$\{\mathit{mk}\text{-}\Sigma(0, 0), \mathit{mk}\text{-}\Sigma(1, 0), \mathit{mk}\text{-}\Sigma(1, 1)\}$$

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

$$\begin{aligned} & \mathit{choose\_b} \\ & \mathsf{ext\ rd}\ b : \mathbb{N} \\ & \quad\ \ \mathsf{wr}\ b : \mathbb{N} \\ & \mathsf{pre\ true} \\ & \mathsf{post\ true} \end{aligned}$$

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 chapter 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 4.1 (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 *possibility* of a read-respecting implementation.

Of course it is equally possible that a non-read-respecting implementation will give rise to a read-respecting interpretation. For example, the semantics cannot distinguish the valid coding $b := 0$ above, from many other codings such as $b := a - a$ or $\mathsf{if}\ a = 0\ \mathsf{then}\ b := 0\ \mathsf{else}\ b := 0$ which have exactly the same semantics but syntactically

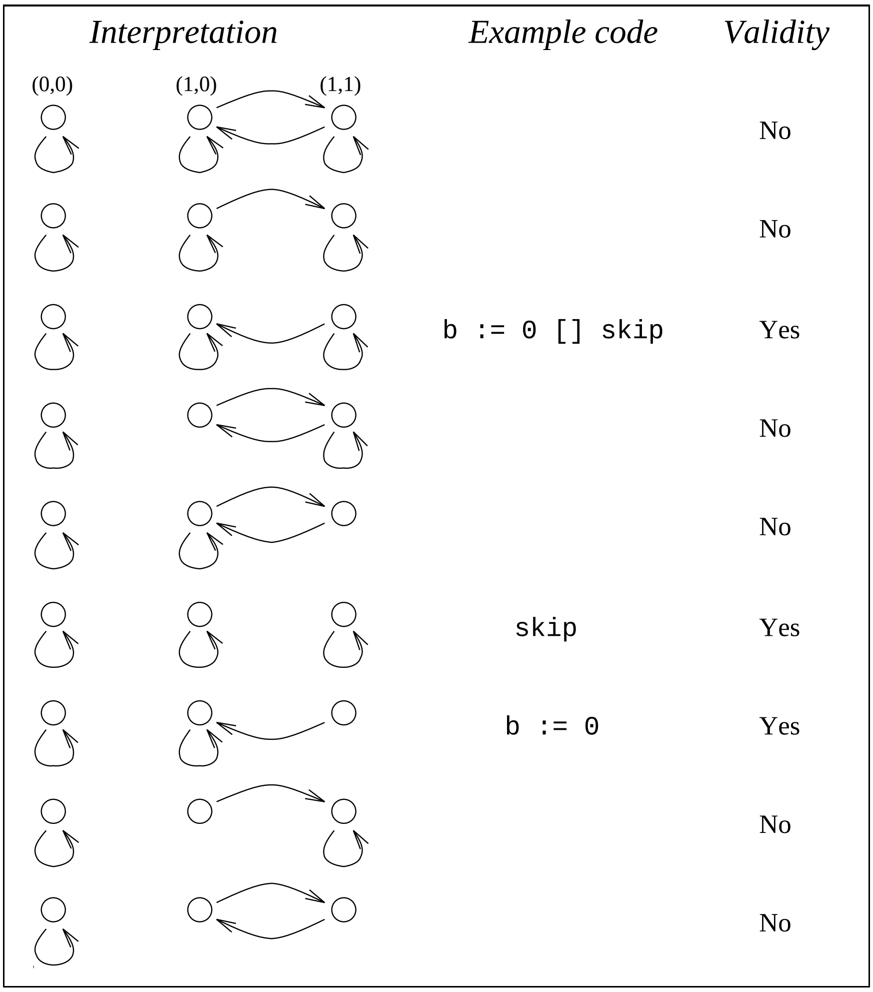
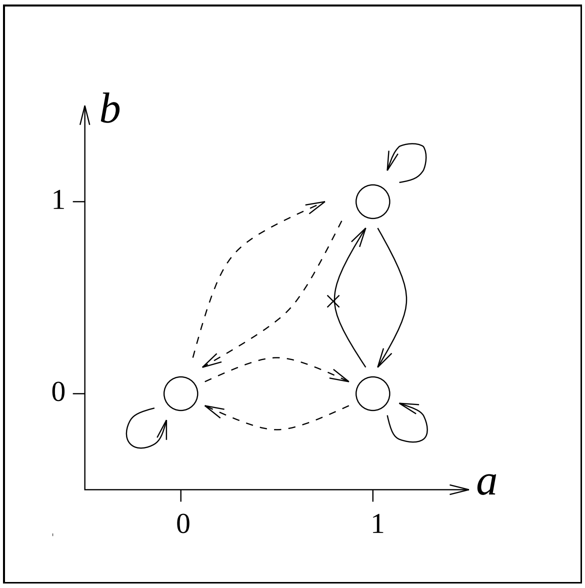



Figure 4.1: All domain preserving subsets of the interpretation of *choose_b*.

do not respect the read frame[3]. The same is true for the interpretation of the write frame. Code, such as $x := x$, may reset a variable to its original value and hence have a write-respecting interpretation, whilst syntactically breaking the write constraint.

To understand the read constraint more fully, it might be helpful to picture the state-space and state-transitions in two dimensions as follows:




The write frame can only be satisfied if there is no transition changing the value of $a$. The read frame requires the removal of 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 frames *can be* 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.

$$\begin{array}{lll} \Sigma :: & a & : \mathbb{N} \\ & b & : \mathbb{N} \\ & c & : \mathbb{N} \end{array}$$

[3]Joe Morris describes this as "opening the door, having a look, and closing the door again".

$inv\text{-}\Sigma(a, b) \quad \triangleq$
$\quad a \in \{0, 1\} \wedge$
$\quad b \leq a \wedge$
$\quad c \leq b$

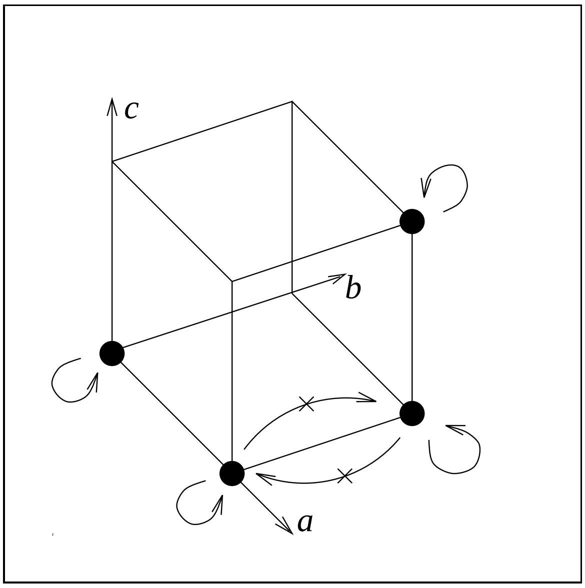
In this case the only possible states are

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

Again consider the operation that can read and write $b$ only

$choose\_b$
ext rd $b$ : $\mathbb{N}$
  wr $b$ : $\mathbb{N}$
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.)

## 4.3 Relational semantics with read and write frames

This section outlines 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.

### 4.3.1 Notation

Recall the notation for hooking and quantification introduced in section 3.2.2.

For $S = \{x_a \mid a \in \alpha\}$ then $\overleftarrow{S} = \{\overleftarrow{x_a} \mid x_a \in S\}$ and for $S_1 \subseteq S$ then

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

Also, for $E\colon Exp(S)$ then $\overleftarrow{E}^{S_1} \triangleq E[\overleftarrow{\sigma_i}/\sigma_i]_{\sigma_i \in S_1}$ thus $\overleftarrow{E}^{S_1}\colon Exp(\overleftarrow{S}^{S_1})$.

If $S$ is a subset of the fields of some composite type then

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

where the $v_i$ are fresh variables and $T_{S_1}$ are the types of the respective fields of the composite type.

Similarly $\forall \overleftarrow{\mathsf{S}} \cdot \overleftarrow{E}$ is a shorthand for the corresponding hooked formula.

**Projection.**

One new notation will be used in this chapter. 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

$$(a, b, c, d)\Big|_{\{b,d\}} \;\triangleq\; (b, d)$$

Thus we have[4]

$$\overleftarrow{\sigma_i}^{S} = (\overleftarrow{\sigma_i}\Big|_{S}, \sigma_i\Big|_{F-S})$$

[4]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. We also follow the VDM convention of using variable names corresponding to the field names.

### 4.3.2 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 $\Sigma$ 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,

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

$\Sigma \xrightarrow{\mathrm{RW}} \Sigma \subseteq \Sigma \times \Sigma$ 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 respectively $\overleftarrow{\sigma} \xrightarrow{\mathrm{W}} \sigma$ and $\overleftarrow{\sigma} \xrightarrow{\mathrm{RW}} \sigma$. Note that, although single headed arrows are used in the relations above, none of these relations are necessarily functions.

Thus we have

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

where

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

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.

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

where

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

The task now remaining is to define an extra constraint on the semantics which interprets the constraint imposed by the read frame. However, this constraint is not simply a predicate on individual transitions but rather depends on the whole set of transitions. The required condition is explored in the following sections and then treated more formally in the next chapter.

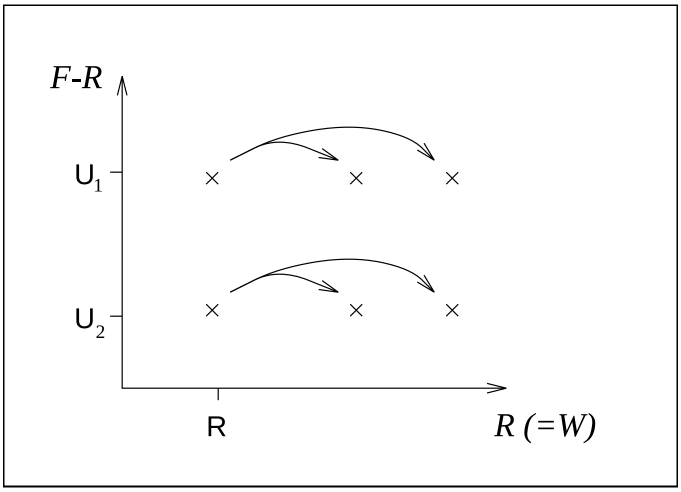
### 4.3.3 Simplification

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 the 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 $\Sigma \xrightarrow{\mathrm{RW}} \Sigma$ be the largest subset of $\Sigma \xrightarrow{\mathrm{W}} \Sigma$ such that

$$\forall \overleftarrow{\mathsf{R}}, \overleftarrow{\mathsf{U}_1}, \overleftarrow{\mathsf{U}_2} \cdot \left\{ \mathsf{R}_1 \;\middle|\; (\overleftarrow{\mathsf{R}}, \overleftarrow{\mathsf{U}_1}) \xrightarrow{\mathrm{W}} (\mathsf{R}_1, \overleftarrow{\mathsf{U}_1}) \right\} = \left\{ \mathsf{R}_2 \;\middle|\; (\overleftarrow{\mathsf{R}}, \overleftarrow{\mathsf{U}_2}) \xrightarrow{\mathrm{W}} (\mathsf{R}_2, \overleftarrow{\mathsf{U}_2}) \right\}$$

which can be reexpressed as

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \Rightarrow \left\{ \sigma_1\Big|_R \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma_1}^{F-R} \right\} = \left\{ \sigma_2\Big|_R \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma_2}^{F-R} \right\}$$

But now note $R = W$ and that $\overleftarrow{\sigma} \xrightarrow{\mathrm{W}} \sigma \;\Rightarrow\; \overleftarrow{\sigma}\Big|_{F\text{-}W} = \sigma\Big|_{F\text{-}W}$ so we have

$$\left\{ \sigma\Big|_R \;\middle|\; \overleftarrow{\sigma} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma}^{F\text{-}R} \right\} = \left\{ \sigma\Big|_R \;\middle|\; \overleftarrow{\sigma} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma}^{F\text{-}W} \right\} = \left\{ \sigma\Big|_R \;\middle|\; \overleftarrow{\sigma} \xrightarrow{\mathrm{W}} \sigma \right\}$$

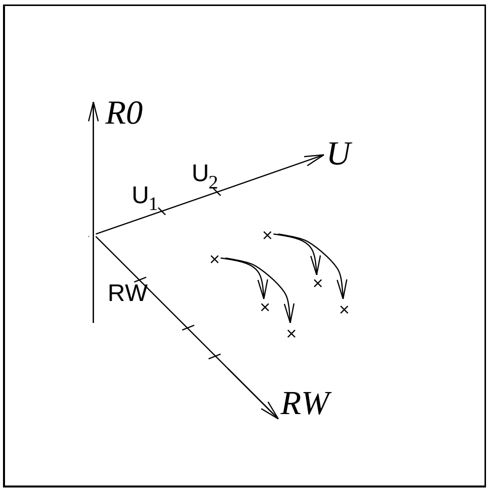
and so the above can be written as

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \Rightarrow \left\{ \sigma_1\Big|_R \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1 \right\} = \left\{ \sigma_2\Big|_R \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2 \right\} \tag{4.1}$$

### 4.3.4 Generalisation

Now consider the case where $W \subseteq R \subseteq F$. In actual fact it is really quite similar. The $R$ components must be split into 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 defined by the read-only variables. Each horizontal plane satisfies the constraints of the previous section, but there is not necessarily any correspondence between planes.

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

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

Which again can be reexpressed

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \Rightarrow \left\{ \sigma_1\Big|_W \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma_1}^{F-W} \right\} = \left\{ \sigma_2\Big|_W \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \overleftarrow{\sigma_2}^{F-W} \right\}$$

Note the judicious use of $R$s and $W$s in the subscripts and superscripts.

Now since

$$\sigma_2\Big|_{R-W} = \overleftarrow{\sigma_2}\Big|_{R-W} = \overleftarrow{\sigma_1}\Big|_{R-W} = \sigma_1\Big|_{R-W}$$

we can replace the $W$s in the subscripts (and superscripts) by $R$s and again reduce this predicate to the one given for the case $R = W$ (4.1).

Note that $W$ only appears in this definition in its role defining the relation $\Sigma \xrightarrow{\mathrm{W}} \Sigma$. This suggests that it may be possible to handle the read and write frames separately, defining two relations each respecting one frame but not the other and then combining the two to give an interpretation respecting both frames. This idea is followed up in the next chapter but first we close this chapter by considering some consequences of this interpretation of the read frame.

### 4.3.5 Satisfiability

We now show that with this read frame constraint, the modified definition of satisfiability given in chapter 3 is equivalent to the usual one.

Recall the standard definition of satisfiability (not respecting the read frame) which can be written

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

and the modified definition of satisfiability respecting frames

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

Under the present semantics we have the following extra property

$$\begin{aligned}(3) \qquad \forall \overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}, \overleftarrow{\mathsf{U}_2} \cdot \; & \left\{ \mathsf{RW}_1 \;\middle|\; (\overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}) \xrightarrow{\mathrm{w}} (\mathsf{RW}_1, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_1}) \right\} \\ = \; & \left\{ \mathsf{RW}_2 \;\middle|\; (\overleftarrow{\mathsf{RW}}, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_2}) \xrightarrow{\mathrm{w}} (\mathsf{RW}_2, \overleftarrow{\mathsf{RO}}, \overleftarrow{\mathsf{U}_2}) \right\}\end{aligned}$$

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 that if for a particular $\overleftarrow{\mathsf{R}}$ and $\overleftarrow{\mathsf{U}_1}$ there is a suitable $\mathsf{W}$, then by (3), the same $\mathsf{W}$ suffices for $\overleftarrow{\mathsf{R}}$ and $\overleftarrow{\mathsf{U}_2}$, which is (2).

## 4.4 Non-interference

As described in Chapter 1, a 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[5]

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

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

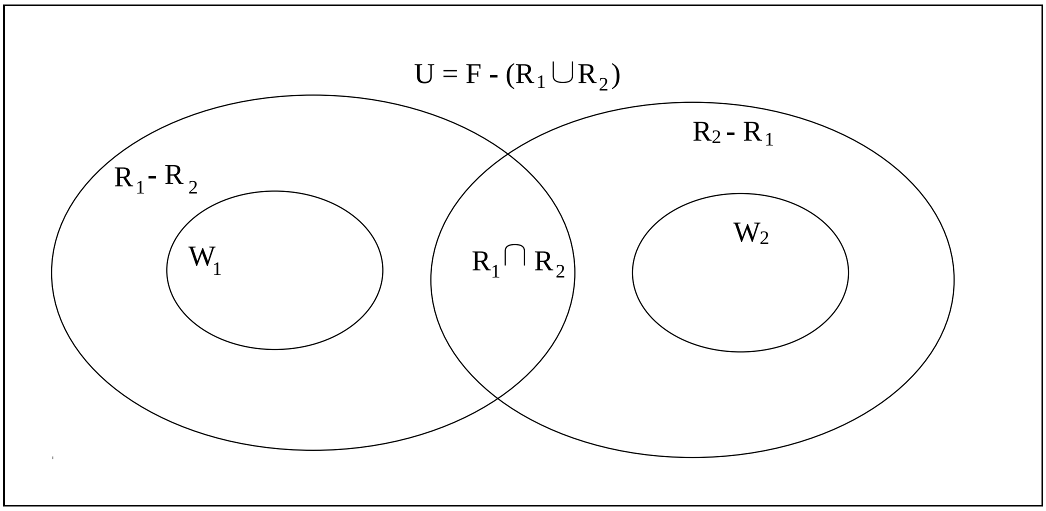
(2) $Op'_1;Op'_2 = Op'_2;Op'_1$

The proof of this is deferred to Section 6.10. Here we give an informal indication of how it proceeds for (1).

We need to show that

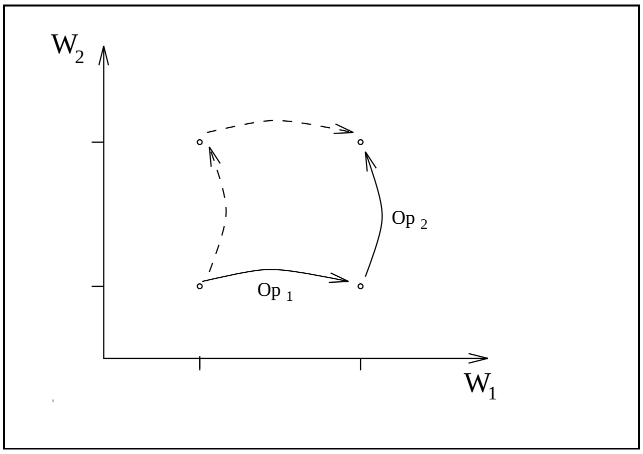
$$[\![Op_1;Op_2]\!] = [\![Op_2;Op_1]\!]$$

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. Such a transition must clearly be the composition of one transition solely in the $W_1$ direction and another in the $W_2$ direction:

[5] 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.




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 frame 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.

## 4.5 Discussion: Read frames and satisfaction

This chapter has outlined a denotational semantics for operations that gives an 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 4.2. 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 4.3.2, 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. 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 from Section 4.3.2, this approach is perfectly adequate since for any operation, the interpretation of any refinement is a refinement of the interpretation. For the read-frame-respecting interpretations, on the other hand, 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 4.2, 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 "no read" condition does not automatically carry down to all refinements of an interpretation. That is, not all refinements of an interpretation satisfying the read frame constraint, necessarily satisfy the constraint themselves. Thus, it is not the case that the information in the frame is being captured in its entirity 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.

This supports the observation made in Chapter 1 that read frames are not preserved by refinement. On the other hand, the proof rules in Chapter 3 do characterise refinements which respect the read frame. Now we see that the usual semantic model is insufficiently rich to support these refinement rules even when read frames are considered. 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. The latter approach is the subject of the investigations in Chapter 6. Prior to that, Chapter 5 prepares the ground by reworking the standard semantic model and incorporating the ideas of this chapter.

Before we proceed, however, let us pause for a moment to consider the route that is being taken. As we are clearly moving towards a more complex semantic model of operations, it is reasonable to ask whether the extra complexity has any payoff in terms of the practical formal development of software. This question can be addressed on several levels. Firstly one could argue that the extra complexity in the semantic model of operations will be counterbalanced by a simplification of the refinement relation. However such a "neutral" advantage is hardly a reason to rework established foundations. Secondly, one might say that it is encumbent on a putative *formal* method to have all syntactic devices for reasoning backed up by a fully formal semantic model. This is indeed fair comment, however, an alternative approach would be to remove the syntactic device in question and survive with impoverished theory for reasoning. But finally, perhaps the most relevant argument is that in fact there is no extra complexity introduced in practise. The proposed 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. If such rules were not available, then our informal understanding would work against the natural construction of justifiable designs.

We have already seen the rules in question presented in Chapter 3, where they were justified by an informal understanding of the domain of discourse. Such an informal understanding, backed up by formal rules of reasoning to validate intuition, is the usual *modus operandi* of an engineering discipline. Model theory and formal semantics are far more the realm of the scientist than the practising engineer. On the other hand, 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 undesirable for a supposedly formal notation. As Wittgenstein concludes in his *Tractatus Logico-Philosophicus*[Wit22]

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

# Chapter 5

# Operation Semantics with Read Frames

*"There are then two mathematics."*

## 5.1 Introduction

We have seen how, although it carries a strong informal understanding, the read frame is not formalised by the usual semantics of operations. The last chapter introduced a means by which we can interpret the *possibility* of a read respecting implementation, this chapter will formalise that understanding and investigate how it interacts with the interpretation of the other parts of the operation. In particular, the treatment of read and write frames will be such that each can be considered in isolation of the other.

However, as discussed at the end of the last chapter, this approach still does not give a satisfactory treatment of read frames in the presence of refinement. The semantic model developed in this chapter will be extended in the next in order to address refinement and this intention is borne in mind in the definitions given here.

### 5.1.1 Recap

As described in Chapter 4, Jones' work [Jon87] gives a semantics for total correctness in the presence of non-determinism via a semantic model for operations consisting of a termination set and meaning relation

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

When looseness is present in the operation definition, the semantics does not distinguish between looseness intended to indicate non-determinism and looseness indicating under-specification. The same is true in VDM-SL [ISO93] which interprets operations as non-deterministic and functions as underspecified.

Read and write frames are not considered in the semantics of [Jon87] except that the write frame justifies a "rest unchanged" assertion. The read frame is seen merely as providing a binding for the variables in the precondition and postcondition rather than a constraint on the implementing code.

A similar aproach is taken in Morgan's Refinement Calculus[Mor90], where write frames indicate unchanged variables and there is no explicit read frame. In RAISE [RLG92, GP92], the read and write frames do play a role in the proof theory as syntactic constraints on the assignments allowable by implementations. In B [Abr95], they also play a role as syntactic constraints. By writing operations as generalised substitutions, it is syntactically clear which variables are to be assigned new values by the operation. This approach is also taken in Back's work(e.g. [Bac88]).

For a given operation with read frame $R$ and write frame $W$, Chapter 4 defines three meaning relations, $\Sigma \xrightarrow{\circ} \Sigma$, which ignores both read and write frames, $\Sigma \xrightarrow{\mathrm{W}} \Sigma$, which only takes account of the write frame and ignores the read frame, and $\Sigma \xrightarrow{\mathrm{RW}} \Sigma$ which respects both frames. Recall that the read frame is interpreted by the requirement that the values of unread variables cannot affect the valid resulting states. This is formalised by insisting that $\Sigma \xrightarrow{\mathrm{RW}} \Sigma$ is the largest subset of $\Sigma \xrightarrow{\mathrm{W}} \Sigma$ such that the following criterion holds:

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow\; \left\{\sigma_1\Big|_R \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1\right\} = \left\{\sigma_2\Big|_R \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2\right\} \tag{4.1}$$

### 5.1.2 Notation

In addition to the notation in Sections 3.2.2 and 4.3.1, it will be convenient to make the following definitions.

For a predicate $P$ on some set $\Sigma$, write the characteristic set of $P$ as $\overline{P}$.

$$\overline{P} = \{s \in \Sigma \mid P(s)\}$$

Define

$$\mathcal{D}_o = \mathcal{P}\Sigma \times \mathcal{P}(\Sigma \times \Sigma)$$

As in previous chapters, here $\Sigma$ is used to denote the state space spanned by the variables in scope, unrestricted by the invariant. Where the restricted state space is required, it will be written as $\Sigma \dagger I$.

We will use the following variable names:

$\sigma: \Sigma$

$T: \mathcal{P}\Sigma$

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

$(T, M): \mathcal{D}_o$

and derivations of them as appropriate.

### 5.1.3 Overview of chapter

The next section, (5.2), reconsiders the interpretation of read frames in order to allow a treatment which is independant of the write frames and Section 5.3 reformulates this definition in a more concise form. Section 5.4 considers the effect of the read frame on the termination set and Section 5.5 the interpretation of preconditions and postconditions. Finally, Section 5.6 brings all these concerns together to define a sematic model for operations.

## 5.2 Orthogonalising the treatment of read and write frames

The approach taken thus far has been to treat the interpretation of the read frame as an extra constraint on the meaning relation over and above the constraints imposed by the predicates (the invariant, precondition and postcondition) and the write frame. This section redefines the read frame constraint in a manner which makes the treatment of read and write frames more orthogonal.

The aim is to define two meaning relations, each respecting one frame and ignoring the other, such that the interpretation respecting both frames can be recovered as some combination of the two. The first interpretation, $\Sigma \xrightarrow{\mathrm{W}} \Sigma$, which respects the write frame (only) has already been defined. The second, respecting the read frame (only), written $\Sigma \xrightarrow{\mathrm{R}} \Sigma$, will be defined in this section. As might have been expected, the construction which combines the two is intersection and thus we seek to show

$$\Sigma \xrightarrow{\mathrm{RW}} \Sigma = (\Sigma \xrightarrow{\mathrm{W}} \Sigma) \cap (\Sigma \xrightarrow{\mathrm{R}} \Sigma)$$

For a given operation, the interpretation $\Sigma \xrightarrow{\mathrm{R}} \Sigma$, which respects the read frame but ignores the write frame, must be equivalent to a read-write respecting interpretation of some operation with the same predicates and read frame, but with a universal write frame. Clearly, this operation may have a write frame which is bigger than its read frame and so, to make this definition, we will need to give an interpretation to variables which are in the write frame but not the read frame.

But how are we to make sense of such "write-only" variables? Although such variables are not normally used in VDM operations, and arguably do not correspond to any useful programming concept, perhaps such unrealistic constructs, much like miracles in [Mor88a], can play a *part* in defining useful operations and so help us to understand the underlying concepts.

The following subsections propose three alternative interpretations for write-only variables and discuss the merits of each. These alternatives will become relevant again in

Section 7.2. We begin by considering the strictest of the three interpretations.

### 5.2.1 Must-write semantics

The first interpretation we consider for variables in the write frame but not the read frame is that they *must* be written by the operation with a value which depends only on the reads.

To understand the motivation for this interpretation, let us consider again the example from previous chapters.

*choose_b*
ext rd $b$
    wr $b$
post true

Clearly, this operation can be implemented by skip, but what are its other possible implementations? The simplest implementations can be divided into two forms depending on whether they read the old value of $b$:

$b := e$ where $e$ is an expression in $b$, and

$b := c$ where $c$ is a constant expression not refering to any variables.

In this light, one might think of skip, which after all is akin to $b := b$, as being of the first of these forms. Thus we think of skip as being parameterised by $b$.

So what should the above operation mean if the read frame were empty? In this case, its implementations would be of the second kind only. That is, they *must* assign a constant to $b$. Thus $\mathsf{skip}_b$ (which is equivalent to $b := b$) should not be a valid implementation and so we call this interpretation of write-only variables a "must-write" semantics[1].

Note that this operation, with write frame $\{b\}$, empty read frame and trivial predicates, specifies the class of implementations which set the value of $b$ to a constant. Interestingly, this class of implementations cannot be characterised in a 'straight' specification.

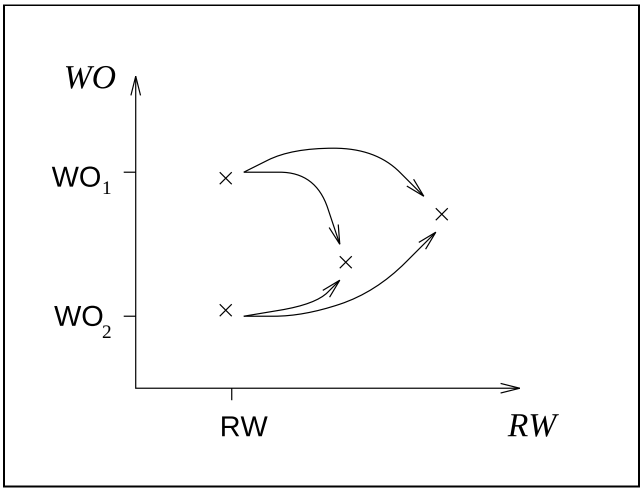
**Variable initialisation.** This interpretation could be used to formalise the initialisation of state and local variables since, with this semantics, write-only variables can be used to specify that state initialisation should obliterate previous data[2], or that local variables should be initialised before use. With this interpretation, the original values

[1]Note that by disallowing skip as an implementation, we may render this operation infeasible: any implementation must ensure that the invariant holds with the constant assigned to the variable $b$, whatever the values of the other variables may be.

[2]For VDM, initialisation is currently formalised as a predicate on states rather than as an operation. An alternative would be to model initialisation as a 'write-only' operation. In earlier versions of

of the write-only components can have no effect whatsoever on the possible final states. Even the final values of the write only variables themselves must be determined from only the read variables, thus it is not possible to leave them unchanged.

To formalise this definition, first consider a state with universal write frame and read frame $R$. In this case there are two sets of variables, one set is read-write and the other is write-only. In this interpretation, we require that two starting states which differ only in the $WO$ component must have exactly the same possible final states - not just the same $RW$ components. That is, any result state possible from a given start state should also be possible from any other start state that differs only in a write-only component:




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

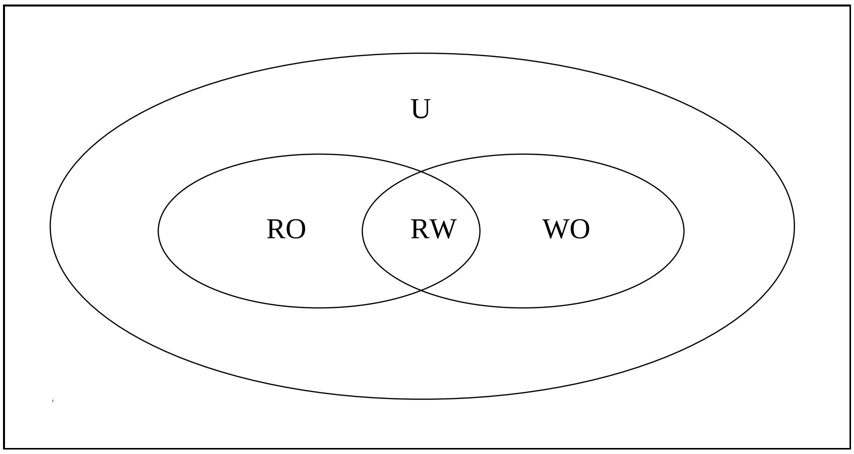
which, as in Chapter 4, can also be rewritten

$$\forall \overleftarrow{\sigma}_1, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow\; \left\{\sigma_1 \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1\right\} = \left\{\sigma_2 \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2\right\} \tag{5.1}$$

Although similar in form to (4.1), this formulation is stronger since there is no projection in the bound variable of the set comprehension.

When this idea is combined with a non-universal write frame, the state splits into four sets of fields. Let $U$ be unread and unwritten, $RO$ be read only, $WO$ be write-only and $RW$ be the read-write part:

VDM [Jon80], and in B, initialisation is given as an operation with syntactic constraints ensuring that it is indeed a write only operation, however these constraints are not formalised in the semantics.




Here, the transition diagram is best 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 like the one above. Planes separated by a change in $U$ need to satisfy the read frame condition from Section 4.3.3. Planes separated by a change in $RO$ need have no correspondence.

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

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

This says that for starting states with the 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, any difference there may have been originally should be ignored. Thus the old values of $WO$ components can have no effect on the possible outcomes.

This can be re-expressed as a quantification over states and in fact yields the same formula as the one *first* given in the last chapter for the case $R \supseteq W$

$$\forall \overleftarrow{\sigma}_1, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \Rightarrow \left\{\sigma_1\Big|_W \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\text{W}} \overleftarrow{\sigma_1}^{F-W}\right\} = \left\{\sigma_2\Big|_W \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\text{W}} \overleftarrow{\sigma_2}^{F-W}\right\}$$

Thus to demonstrate the orthogonality which is sought, the condition (5.1) must reduce to this when we introduce the "rest-unchanged" constraint imposed by the write frame.

However, (5.1) forces equality on the whole after states and so, in particular, on those components which are neither read nor written

$$\sigma_1\Big|_{F-(R\cup W)} = \sigma_2\Big|_{F-(R\cup W)}$$

whereas the write frame requires identity on the unwritten components which includes those which are also unread

$$\sigma_i\Big|_{F-(R\cup W)} = \overleftarrow{\sigma}_i\Big|_{F-(R\cup W)}$$

so we see that we have a contradiction in the case where the initial states differ on the unreferenced components

$$\overleftarrow{\sigma}_1\Big|_{F-(R\cup W)} \neq \overleftarrow{\sigma}_2\Big|_{F-(R\cup W)}$$

Thus this interpretation of write-only variables is too strong for the orthogonality we seek.

### 5.2.2 May-write semantics

A second, slightly more liberal, alternative arises if we reintroduce the possibility of skip on the write-only variables. This corresponds more closely to what might be expected to arise from a static analysis of code ascertaining which variables are read and written. By allowing skip on unread variables, this interpretation would allow their final values to be the same as their initial values, but it would not allow their original values to affect the final values in any other case. Thus we call this a *may*-write semantics.

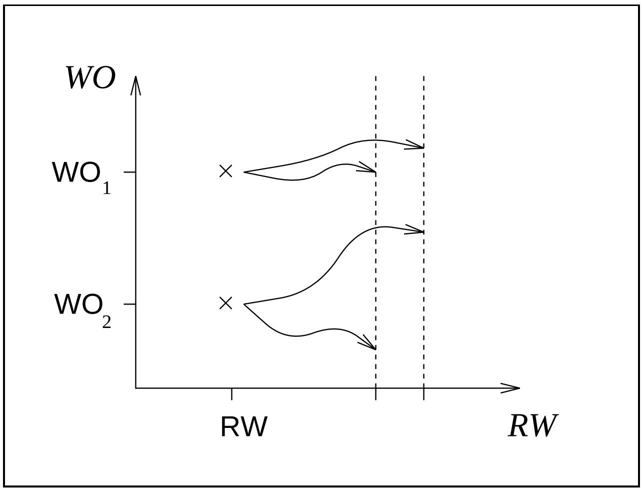
This semantics can be formalised by explicitly adding to the must-write semantics the possibility of no change of value for the unread variables. However, in this case, the formulation is rather unwieldy and its precise definition is deferred to Section 7.2.2 when more notation will be available. For the present we simply observe that for any $\overleftarrow{\sigma_1}$ and $\overleftarrow{\sigma_2}$ which are equal on the read variables, and for each write only variable, $x$, we must allow skip on $x$ from $\overleftarrow{\sigma_1}$ if and only if it is allowed from $\overleftarrow{\sigma_2}$. Furthermore, any other result value possible for $x$ from $\overleftarrow{\sigma_1}$ must also be possible from $\overleftarrow{\sigma_2}$ and vice versa. Note, that a change in $x$ from $\overleftarrow{\sigma_1}$ could result in the value of $x$ in $\overleftarrow{\sigma_2}$, so care must be taken when bringing these two conditions together.

### 5.2.3 Secure statements

The third and most liberal alternative, not only allows skip on unread variables but also allows some other "secure" statements. Such statements are those that, whilst allowing the value of the unread variables to be changed, do not allow information as to what those values are to "leak" into the value of the read variables. Thus, the original values of the write-only variables could influence their own final values, and those of other write-only variables, but not the final values of any read variable. Examples of such statements include statements to increment a number, append to, or reverse a list, or even to swap the values in two variables which are both write-only.

This is clearly an information flow condition, indeed it very similar to the "no read up and no write down" constraint in the Bell and LaPadula Model for confidentiality [BL76] and thus may have some applications in the security domain. For this reason it is called a 'secure' semantics.

The condition required to yield this interpretation states that the initial values of write-only variables should be able to influence their own final values and those of any other write-only variables, but should not influence the final values of any read variables. This is formalised by weakening the predicate given above to require merely the existence of some state with the required read component. For an operation with universal write frame, this can be pictured as




Here we see that it would be impossible to deduce the initial values of the write-only variables from the final values of only the read variables.

This interpretation gives a predicate which is the same as the *final* one given for the case $R \supseteq W$ in the last chapter (4.1). Furthermore, since $W$ is universal, we can write $\xrightarrow{\circ}$ in place of $\xrightarrow{\mathrm{W}}$ to give[3]

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow\; \left\{ \sigma_1\Big|_R \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \right\} = \left\{ \sigma_2\Big|_R \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \right\}$$

It is clear that when we reintroduce the 'rest-unchanged' clauses from a write frame inside the reads, the freedom on the unread variables arising from existential quantification is constrained to the one value for which the final values are equal to the initial ones[4]. So this finally yields the orthogonality between the two interpretations that we seek

[3] Note that there is an implicit existential quantification in each set comprehension since the $\sigma_i$ have unbound variables for the unread components.

[4] To see this note that for $x \in W \subseteq R$ we have $\sigma_1\Big|_{\{x\}} = \overleftarrow{\sigma_1}\Big|_{\{x\}} = \overleftarrow{\sigma_2}\Big|_{\{x\}} = \sigma_2\Big|_{\{x\}}$.

$$\Sigma \xrightarrow{\text{RW}} \Sigma = \Sigma \xrightarrow{\text{R}} \Sigma \cap \Sigma \xrightarrow{\text{W}} \Sigma$$

## 5.3 Reformulation of the read constraint

The above constraint on the semantics of operations interprets the intention that variables outside the read frame can have no effect on the outcome of an operation. Furthermore, the constraint involved is independent of the treatment of the write frame. This section re-expresses this constraint in a more concise form.

The predicate in question is

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow\; \left\{ \sigma_1\Big|_R \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \right\} = \left\{ \sigma_2\Big|_R \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \right\}$$

which by simple set theory, can be rewritten as

$$\begin{array}{l} \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow \\ \qquad \forall \sigma_1 \in \left\{ \sigma_1 \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \right\} \cdot \exists \sigma_2 \in \left\{ \sigma_2 \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \right\} \cdot \sigma_1\Big|_R = \sigma_2\Big|_R \\ \quad \wedge \\ \qquad \forall \sigma_2 \in \left\{ \sigma_2 \;\middle|\; \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \right\} \cdot \exists \sigma_1 \in \left\{ \sigma_1 \;\middle|\; \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \right\} \cdot \sigma_1\Big|_R = \sigma_2\Big|_R \end{array}$$

which in turn is equivalent to

$$\begin{array}{l} \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1}\Big|_R = \overleftarrow{\sigma_2}\Big|_R \;\Rightarrow \\ \qquad \forall \sigma_1 \cdot \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \;\Rightarrow\; \exists \sigma_2 \cdot \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \wedge \sigma_1\Big|_R = \sigma_2\Big|_R \\ \quad \wedge \\ \qquad \forall \sigma_2 \cdot \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \;\Rightarrow\; \exists \sigma_1 \cdot \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \wedge \sigma_1\Big|_R = \sigma_2\Big|_R \end{array}$$

Now the reader may recognise this formula as akin to the definition of a (strong) bisimulation from process algebra (eg. [Mil89]). Recall that a relation $\mathcal{S}$ over agents is a (strong) bisimulation if

> "For any agents $P$ and $Q$ such that $(P, Q) \in \mathcal{S}$, and for all actions $\alpha \in Act$ we have:
>
> i) $P \xrightarrow{\alpha} P' \;\Rightarrow\; \exists Q' \cdot Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$, and
> ii) $Q \xrightarrow{\alpha} Q' \;\Rightarrow\; \exists P' \cdot P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$"

Here $P, Q, P', Q'$ correspond to $\overleftarrow{\sigma_1}, \overleftarrow{\sigma_1}, \sigma_1, \sigma_1$ and $(A, B) \in \mathcal{S}$ corresponds to $A\Big|_R = B\Big|_R$.

We borrow a notation from Z and, generalising slightly, define for any set of identifiers $R$, a relation on states

$$\Xi_R : \mathcal{P}(\Sigma \times \Sigma)$$

$$\Xi_R \quad \triangleq \quad \{(\overleftarrow{\sigma}, \sigma) \mid \overleftarrow{\sigma}\big\rfloor_R = \sigma\big\rfloor_R\}$$

This yields the following theorem

**Theorem**

The relation $\Sigma \xrightarrow{R} \Sigma$ is the largest subrelation of $\Sigma \xrightarrow{\circ} \Sigma$ such that $\Xi_R$ is a strong bisimulation on $\Sigma \xrightarrow{R} \Sigma$.

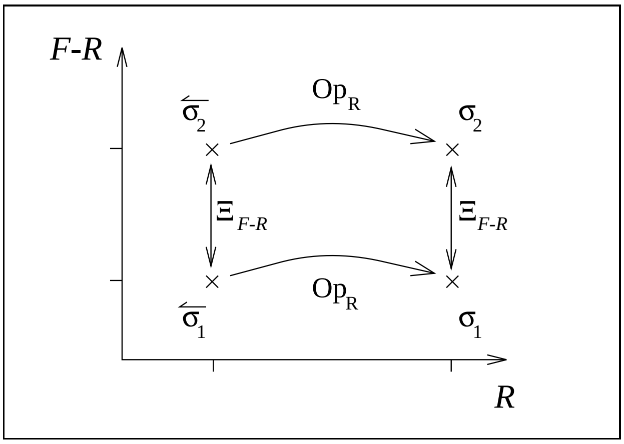
The fact that $\Sigma \xrightarrow{R} \Sigma$ is the *largest* subset of $\Sigma \xrightarrow{\circ} \Sigma$ clearly corresponds to the definition of the weakest strong bisimulation, the existence of which is discussed in [Mil89].

This surprising result warrants some further investigation to understand its significance more fully.

### 5.3.1 'write_W'

Consider that $\Xi_R$ is the meaning of the operation $write_{F-R}$ which is the most general operation that respects the write frame $F$-$R$ but has universal read frame. Clearly, the above theorem indicates some relationship between any operation that respects a read frame $R$ and $write_{F-R}$. This section makes an investigation into that relationship.

Note that for any operation $Op_R$ with read frame $R$, since $Op_R$ does not read the values of variables in $F-R$, and $write_{F-R}$ only affects those variables, whatever behaviour was possible for $Op_R$ from a given state, will also be possible if preceded by the execution of $write_{F-R}$. The resulting states in the two cases again being equal on $F-R$.



This is formalised by the condition[5]

$$[\![ write_{F-R} ; Op_R ]\!] \subseteq [\![ Op_R ; write_{F-R} ]\!]$$

which can be shown as follows. Let

$$(\overleftarrow{\sigma_1}, \sigma_2) \in [\![ write_{F\text{-}R} ; Op_R ]\!]$$

So there exists some $\overleftarrow{\sigma_2}$ such that

$$(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}) \in [\![ write_{F\text{-}R} ]\!] \quad \text{and} \quad (\overleftarrow{\sigma_2}, \sigma_2) \in [\![ Op_R ]\!]$$

Therefore, by the read constraint on $Op_R$, there exists some $\sigma_1$ such that

$$(\overleftarrow{\sigma_1}, \sigma_1) \in [\![ Op_R ]\!] \wedge \sigma_1 \Big|_R = \sigma_2 \Big|_R$$

thus we have

$$(\overleftarrow{\sigma_1}, \sigma_2) \in [\![ Op_R ; write_{F\text{-}R} ]\!]$$

as required.

However, one should not be mislead into believing that this containment is in fact an equality, for in the presence of an invariant the necessary intermediate state may not exist. To see this, consider the now familiar example from Chapters 3 and 4

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

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

Define

*write_b*

ext rd $a, b : \mathbb{N}$
    wr $b \quad : \mathbb{N}$

pre true

post true

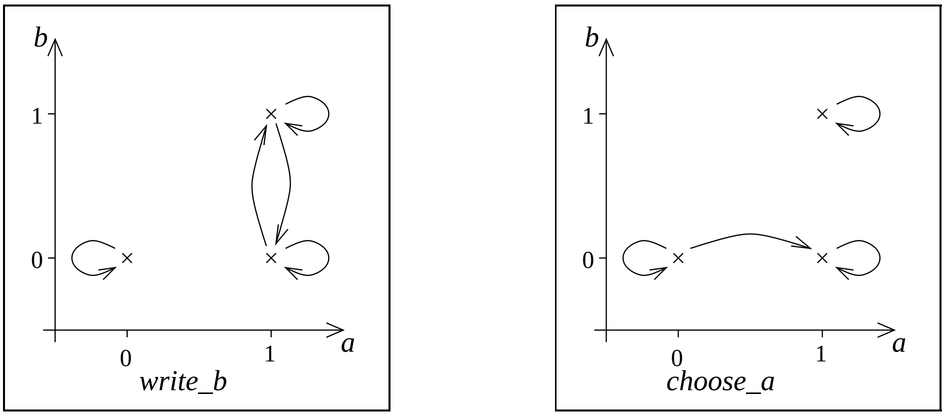
*choose_a*

ext rd $a \ : \mathbb{N}$
    wr $a : \mathbb{N}$

pre true

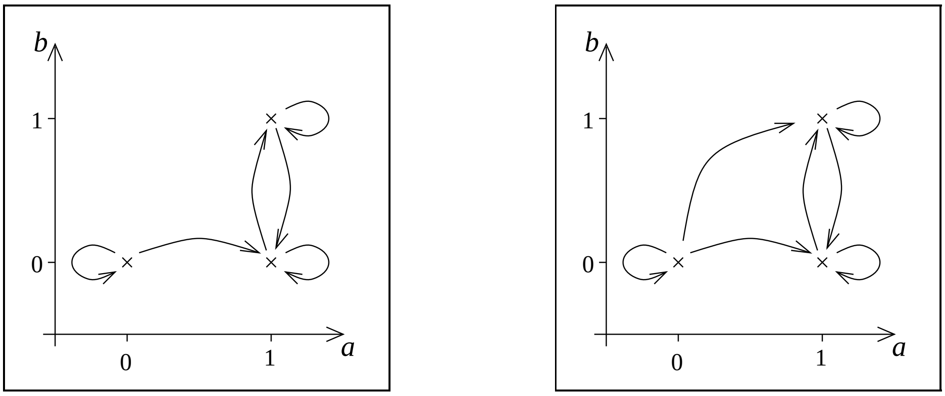
post true

[5]Strictly speaking, the semantic model is a $(T, M)$ pair but, for the present, we are ignoring the termination set.




Note that in *choose_a* the transition $((1,0) \mapsto (0,0))$ is prohibited by the non-reading of $b$.

Now, by the read constraint on *choose_a*, we have




$$\llbracket write_b \; ; \; choose\_a \rrbracket \qquad \subseteq \qquad \llbracket choose\_a \; ; \; write_b \rrbracket$$

but we do not have the opposite containment since

$$((0,0) \mapsto (1,1)) \in \llbracket choose\_a ; write_b \rrbracket$$

whereas

$$((0,0) \mapsto (1,1)) \notin \llbracket write_b ; choose\_a \rrbracket$$

Note that if we had ignored the read frame altogether, that is if we considered the two sequences of operations $write_a ; write_b$ and $write_b ; write_a$, then we would have found that neither contained the other since the former contains $((0,0) \mapsto (1,1))$ but not $((1,1) \mapsto (0,0))$ and vice versa for the latter. Thus we see that it is the read restriction on *choose_a* that introduces the containment in one direction. Note also, that if we introduce the read constraint on both operations we get

$$\llbracket choose\_a ; choose\_b \rrbracket = \llbracket choose\_b ; choose\_a \rrbracket$$

which is an example of the non-interference theorem from Chapter 4.

On the other hand, when not considering the read frames, it is the invariant that can be seen as the source of the incomparability as it proscribes transitions through the 'non-state' $(0, 1)$. Thus, this result is a consequence of the way the invariant is interpreted as part of the state type rather than as an 'extra clause' in the precondition and postcondition. Conversely, when the invariant has no influence, that is when $R \overset{ind}{\subseteq} F$, the condition can indeed be strengthened to an equality without effect and then non-interference with *choose_R* characterises read-respecting behaviour. The persistent interpretation of the invariant will be discussed further in Section 7.3.

The above discussion gives some insight into how the correspondence between bisimulation and read frames has come about. Both can be understood in terms of the observations possible through other operations. An operation which does not read certain variables will not interfere with another which only assigns to those variables. This observation was formalised as a containment of relations and the same approach will lead in the next section to a succinct criterion for bisimulation expressed in relational terms.

### 5.3.2 Reformulation

Before closing this section we use the intuition gained above to rework the definition of bisimulation in terms of relational algebra.

We have that a relation $\mathcal{S}$, over agents is a (strong) bisimulation if, for any agents $P$ and $Q$ such that $(P, Q) \in \mathcal{S}$, and for all $\alpha \in Act$ we have:

i) $P \xrightarrow{\alpha} P' \;\Rightarrow\; \exists Q' \cdot Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$, and
ii) $Q \xrightarrow{\alpha} Q' \;\Rightarrow\; \exists P' \cdot P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$

Now writing $M_\alpha$ for the relation $\{(P, P') \mid P \xrightarrow{\alpha} P'\}$, and writing $\mathcal{S}$ and $M_\alpha$ infixed, the definition of bisimulation can be written as

$$\begin{array}{lll} \forall \alpha \in Act \cdot \forall P, Q \cdot P\mathcal{S}Q & \Rightarrow & (\forall P' \cdot PM_\alpha P' \;\Rightarrow\; \exists Q' \cdot QM_\alpha Q' \wedge P'\mathcal{S}Q') \\ & & \wedge \\ & & (\forall Q' \cdot QM_\alpha Q' \;\Rightarrow\; \exists P' \cdot PM_\alpha P' \wedge P'\mathcal{S}Q') \end{array}$$

Which is equivalent to

$$\begin{array}{ll} \forall \alpha \in Act \cdot & (\forall P, Q, P' \cdot P\mathcal{S}Q \wedge PM_\alpha P' \;\Rightarrow\; \exists Q' \cdot QM_\alpha Q' \wedge P'\mathcal{S}Q') \\ & \wedge \\ & (\forall P, Q, Q' \cdot P\mathcal{S}Q \wedge QM_\alpha Q' \;\Rightarrow\; \exists P' \cdot PM_\alpha P' \wedge P'\mathcal{S}Q') \end{array}$$

Which is equivalent to[6]

$$\begin{array}{ll} \forall \alpha \in Act \cdot & (\forall Q, P' \cdot (\exists P \cdot P\mathcal{S}Q \wedge PM_\alpha P') \Rightarrow (\exists Q' \cdot QM_\alpha Q' \wedge P'\mathcal{S}Q')) \\ & \wedge \\ & (\forall P, Q' \cdot (\exists Q \cdot P\mathcal{S}Q \wedge QM_\alpha Q') \Rightarrow (\exists P' \cdot PM_\alpha P' \wedge P'\mathcal{S}Q')) \end{array}$$

[6] since $(\forall x \cdot (P(x) \;\Rightarrow\; Q)) \;\Leftrightarrow\; ((\exists x \cdot P(x)) \;\Rightarrow\; Q)$, provided $x$ is not free in $Q$

Which is equivalent to

$$\begin{array}{ll} \forall \alpha \in Act \cdot & (\forall Q, P' \cdot Q(\mathcal{S}^{-1};M_\alpha)P' \;\Rightarrow\; Q(M_\alpha;\mathcal{S}^{-1})P') \\ & \wedge \\ & (\forall P, Q' \cdot P(\mathcal{S};M_\alpha)Q' \;\Rightarrow\; P(M_\alpha;\mathcal{S})Q') \end{array}$$

Which is equivalent to the evidently more concise form

$$\forall \alpha \in Act \cdot \; S^{-1};M_\alpha \subseteq M_\alpha;S^{-1} \quad \wedge \quad S;M_\alpha \subseteq M_\alpha;S$$

Note that if $S$ is symmetric (as for $\Xi_R$), then we have $S = S^{-1}$, so either conjunct can be dropped from the body of the quantification without effect.

In our case, we have only one action defined by $\Sigma \xrightarrow{\circ} \Sigma$. So, writing $M$ for $\Sigma \xrightarrow{\circ} \Sigma$, we finally achieve a characterisation of the condition for not reading which is concise

$$reads(R, M) \;\triangleq\; \Xi_R;M \subseteq M;\Xi_R$$

Thus, we have seen how the correspondence between the model-oriented and process algebraic views of a system discussed in Chapter 1 can give an insight into both camps. This would certainly merit further investigation to see how far we can transfer ideas between disciplines. However, such considerations are considered to fall outside the scope of this thesis.

## 5.4 Termination and read frames

Thus far, we have ignored the termination set in the semantic model. However, if the frames are to be interpreted as advanced information about the state accesses of implementing code, and if this code is to respect the read frame, then this will be reflected in the termination set of the operation.

As might be expected, the criterion for read-respecting for a set is similar, but simpler, than that for a relation. A definition can be made by simply replacing the meaning relation in the condition by the termination set[7].

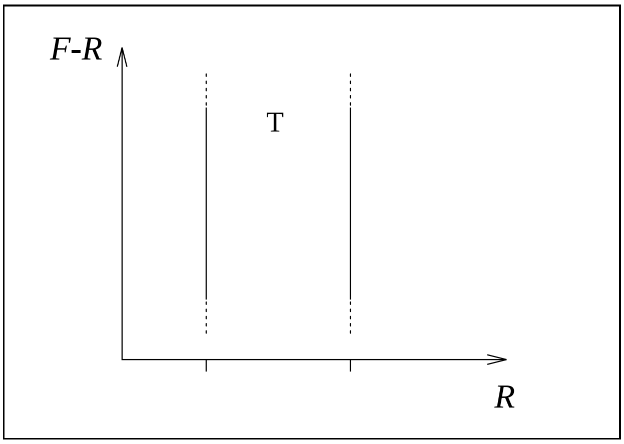
$$reads(R, T) \;\triangleq\; \Xi_R;T \subseteq T;\Xi_R$$

Since $\Xi_R$ is symmetric, the containment can be replaced by an equality without effect. An equivalent, and more immediate, form can be given as a relational image

$$reads(R, T) \;\Leftrightarrow\; \Xi_R \,(\!| T |\!) \subseteq T$$

These conditons amount to saying that a set $T$ respects the frame $R$ if it is a cylinder in the state space.

[7] For a relation $R : \mathcal{P}(\Sigma \times \Sigma)$ and a set $S : \mathcal{P}\Sigma$ we have $T;R \triangleq T \lhd R$ and $R;T \triangleq R \rhd T$.




Thus we also have

$$reads(R, T) \;\Leftrightarrow\; \Xi_R \,(\!|\, \Sigma - T \,|\!) \subseteq \Sigma - T$$

The above equivalences are made precise in the following lemma which is proved in Appendix A.2.1.

**lemma**

For a relation $R: \mathcal{P}(\Sigma \times \Sigma)$ and a set $S: \mathcal{P}\Sigma$ we have

$$R;T \subseteq T;R \;\Leftrightarrow\; R\,(\!|\, T \,|\!) \subseteq T \;\Leftrightarrow\; R\,(\!|\, \Sigma - T \,|\!) \subseteq \Sigma - T$$

Furthermore, if $R$ is symmetric, then

$$R;T \subseteq T;R \;\Leftrightarrow\; R;T = T;R$$

and, if $R \supseteq Id_\Sigma$, then

$$R;T \subseteq T;R \;\Leftrightarrow\; R\,(\!|\, T \,|\!) = T$$

Note that this condition is indeed necessary since it is possible for a non-read-respecting operation, that is one with behaviour different from any with read-respecting code, to have a read-respecting implementation. For example, the code[8] $\mathsf{if}\ x := 0\ \mathsf{then\ skip\ else}\ (\mathsf{skip} \,[\!]\, \mathsf{abort})$ may have the same meaning relation as $\mathsf{skip}$ which respects any read frame. However, their different behaviours are distinguished by the non-read-respecting termination set of the first alternative.

## 5.5 Predicates and frames

The previous two sections have considered the interpretation of the read and write frames in terms of their effect on the meaning relation. This section considers the interpretation

[8] $[\![\mathsf{skip}]\!] = (\overline{\mathsf{true}}, Id_F)$, $[\![\mathsf{abort}]\!] = (\overline{\mathsf{false}}, \overline{\mathsf{false}})$, $[\![\mathsf{skip} \,[\!]\, \mathsf{abort}]\!] = (\overline{\mathsf{false}}, Id_F)$, $[\![\,\mathsf{if}\ x = 0\ \mathsf{then\ skip\ else}\ (\mathsf{skip} \,[\!]\, \mathsf{abort})\,]\!] = (\overline{x = 0}, Id_F)$

of the other components of the operation definition, namely the precondition, invariant and postcondition.

### 5.5.1 Preconditions and feasibility

The precondition in VDM is used to record explicitly the states from which the specified behaviour is required. Were an operation to be executed from a state outside the precondition, then its behaviour would be unspecified, not even termination being guaranteed. The precondition is interpreted in the relational semantics of [Jon87] through an explicit termination set which is defined alongside the meaning relation. This model, which is equivalent to indicating non-terminations by augmenting the meaning relation with transitions resulting in a "bottom" state, is convenient as it simplifies the definition of the semantics of some statement combinators and also provides a basis for an explicit treatment of feasibility.

Feasibility is described in [Jon87] by the condition $T \subseteq \mathsf{dom}\, M$, which states that wherever termination is required, some result state must be permitted by the meaning relation. Jones assumes atomic statements are feasible and, by judicious choice of termination sets, shows that feasibility is maintained by the statement constructors. Therefore, it is ensured that all code is feasible and attention is restricted to feasible specifications. Morgan [Mor88a] argues for the usefulness of infeasible specifications as part of the development process. Such "miracles" can be guarded and then composed to form useful (and feasible) operations. This brings out the difference between preconditioning and guarding that is stressed in the semantics of B [Abr95][9].

[9] In B and Morgan's work the semantics of preconditions (as a weakest precondition) is

$$wp(\mathsf{pre}\ P\ \mathsf{then}\ S\ \mathsf{end}, \mathcal{A}) \quad \triangleq \quad P \wedge wp(S, \mathcal{A})$$

which highlights the fact that a precondition is a help to feasibility since, where $P$ is false, then $P \wedge wp(S, \mathcal{A})$ is false, whatever the latter may be. (In *wp* semantics, feasibility is defined by $feasible(S) \Leftrightarrow wp(S, \mathsf{false}) = \mathsf{false}$.) By contrast, the weakest precondition for *guards* is

$$wp(\mathsf{guard}\ G\ \mathsf{then}\ S\ \mathsf{end}, \mathcal{A}) \quad \triangleq \quad G \;\Rightarrow\; wp(S, \mathcal{A})$$

which is a hinderence to feasibility since anywhere where the guard is false, the implication is true and so the operation is infeasible. Thus $\mathsf{guard}\ G\ \mathsf{then}\ S\ \mathsf{end}$ is feasible only if $G$ holds universally.

Correspondingly, say the relational semantics of $S$ is given by $(T, M)$, then the relational semantics for $\mathsf{pre}\ P\ \mathsf{then}\ S\ \mathsf{end}$ is

$$T_{\mathsf{pre}\ P\ \mathsf{then}\ S\ \mathsf{end}} = \overline{P} \cap T_S$$
$$M_{\mathsf{pre}\ P\ \mathsf{then}\ S\ \mathsf{end}} = M_S \cup (\overline{\neg P} \times \Sigma)$$

whereas the corresponding relational semantics for guards is

$$T_{\mathsf{guard}\ G\ \mathsf{then}\ S\ \mathsf{end}} = \overline{\neg G} \cup T_S$$
$$M_{\mathsf{guard}\ G\ \mathsf{then}\ S\ \mathsf{end}} = \overline{G} \lhd M_S$$

Hence, for a precondition the statement *might* terminate successfully outside the termination set whereas with a guard it is miraculous (i.e. it must terminate but there is no state in which it can terminate).

For an interpretation to respect the precondition, the termination set must contain the set of states satisfying the precondition. Furthermore, by making the invariant part of the state type, Jones' semantics implicitly assumes that operations and sub-operations will only be executed from states satisfying the invariant. Here, where the invariant is being treated explicitly, this assumption must be made explicit in the definition of termination. Thus, termination need only be guaranteed when the operation is executed from a state satisfying the precondition and invariant

$$\overline{P \wedge I} \subseteq T$$

### 5.5.2 Postconditions

The interpretation of postconditions is straightforward. Any execution of the operation which begins from a state satisfying the precondition and invariant must lead to a state transition satisfying the postcondition and a final state satisfying the invariant

$$\overline{P \wedge I} \lhd M \subseteq \overline{Q \wedge I}$$

### 5.5.3 Preconditions and the read and write frames

We have seen in Sections 4.3 and 5.3 that the read and write frame can be interpreted by the following constraints on the meaning relation

$$M \subseteq \Xi_{F\text{-}W}$$

$$\Xi_R ; M \subseteq M ; \Xi_R$$

However, it is reasonable to ask whether the whole of the meaning relation should be constrained by these predicates, or whether, as for the postcondition, the meaning relation should first be domain restricted to the precondition and invariant. That is, whether $\overline{P \wedge I} \lhd M$ should be used in place of $M$ in the above conditions.

In fact, this choice has no real effect on the behaviour of the operation as it is assumed that the operation will only be executed from states satisfying precondition and invariant. However, as the intention is to model syntactic constraints on code, it is reasonable to expect that such code will respect the frames whatever state the operation is executed from. Consequently, we insist that the whole of the meaning relation is constrained by the read and write conditions[10].

Note that although this interpretation of the reads and writes is consistent with the syntactic constraints on code, it is not sufficient to ensure the persistent interpretation as discussed in Section 3.3. This will lead to a problem for the semantics of refinement in Chapter 6 and will require the further enrichment discussed in Section 7.3.

[10]This choice has the added benefit of allowing orthogonalisation of concerns. As $P$ and $I$ and $Q$ are not mentioned in the conditions for $W$ and $R$, the read and write frames can be dealt with independently of the predicates.

## 5.6 Operation semantics

We have seen that, with the 'secure' interpretation of write-only variables, we can give an orthogonal treatment to the read and write frames, and have discussed the interpretation of the various components of an operation definition. We now use these criteria to define a semantic model for operations.

In order to prepare the ground for the richer semantic model that will be used in the next chapter, we take a slightly unusual route to defining the operation semantics. Rather than defining $T$ and $M$ directly from the components of the operation definition, we consider the $(T, M)$ for all the valid implementations of the operation and take the most general of these. For example, for precondition $P$ and postcondition $Q$, instead of

$$T = \{\sigma \mid P(\sigma)\} = \overline{P}$$

$$M = \{(\overleftarrow{\sigma}, \sigma) \mid Q(\overleftarrow{\sigma}, \sigma)\} = \overline{Q}$$

we consider

$$\mathcal{S}_T = \{T_i \mid T_i \supseteq \overline{P}\}$$

$$\mathcal{S}_M = \{M_i \mid M_i \subseteq \overline{Q}\}$$

and then define

$$T = \bigcap \mathcal{S}_T$$

$$M = \bigcup \mathcal{S}_M$$

### 5.6.1 Componentwise interpretation of operations

We can now make the definition of operation semantics we have been building towards. In the following, we discuss a fixed operation

$$Op = mk\text{-}OpDef(F, I, R, W, P, Q)$$

We identify four separate compliance conditions for a $(T, M)$ pair. They are

**dom** which ensures that the predicates are respected by the termination set

$$dom_{(F,I,R,W,P,Q)}(T, M) \quad \triangleq \quad \overline{P \wedge I} \subseteq T$$

**res** which ensures that the predicates are respected by the meaning relation

$$res_{(F,I,R,W,P,Q)}(T, M) \quad \triangleq \quad \overline{P \wedge I} \lhd M \subseteq \overline{Q \wedge I}$$

**writes** which ensures that the write frame is adhered to

$$writes_{(F,I,R,W,P,Q)}(T, M) \quad \triangleq \quad M \subseteq \Xi_{F-W}$$

and

**reads** which ensures the same for the read frame

$$\begin{array}{lll} reads_{(F,I,R,W,P,Q)}(T, M) & \triangleq & \Xi_R ; M \subseteq M ; \Xi_R \;\; \wedge \\ & & \Xi_R (\!| \, T \, |\!) \subseteq T \end{array}$$

The first and second clauses of *reads* will be called *readsM* and *readsT* respectively.

We will define the semantics of an operation to be the most general $(T, M)$ pair which satisfies these four conditions. First we define the set of all the interpretations satisfying the four conditions

$$\mathcal{S} = \left\{ (T_i, M_i) \colon \mathcal{D}_o \;\middle|\; \begin{array}{ll} & dom_{(F,I,R,W,P,Q)}(T, M) \\ \wedge & res_{(F,I,R,W,P,Q)}(T, M) \\ \wedge & writes_{(F,I,R,W,P,Q)}(T, M) \\ \wedge & reads_{(F,I,R,W,P,Q)}(T, M) \end{array} \right\}$$

Then define[11]

$$\begin{array}{l} [\![ mk\text{-}OpDef(F, I, R, W, P, Q) ]\!]_0 \;\; \triangleq \\ \qquad\qquad \iota (T, M) \in \mathcal{S} \cdot \forall (T_i, M_i) \in \mathcal{S} \cdot T \subseteq T_i \wedge M \supseteq M_i \end{array}$$

The well-definedness of such a $(T, M)$ pair is demonstrated in the next section.

Note that each predicate only involves some of the $F, P, I, Q, R, W$ so we can define the three partial interpretations described in Section 5.2 as follows

$$\begin{array}{lll} \Sigma \xrightarrow{\circ} \Sigma & = & [\![ mk\text{-}OpDef(\; F \;,\; I \;,\; F \;,\; F \;,\; P \;,\; Q \;) ]\!]_0 \\ \Sigma \xrightarrow{\mathrm{W}} \Sigma & = & [\![ mk\text{-}OpDef(\; F \;, \mathsf{true} \,, \; F \;, \; W \;, \mathsf{true} \,, \mathsf{true} \,) ]\!]_0 \\ \Sigma \xrightarrow{\mathrm{R}} \Sigma & = & [\![ mk\text{-}OpDef(\; F \;, \mathsf{true} \,, \; R \;, \; F \;, \mathsf{true} \,, \mathsf{true} \,) ]\!]_0 \end{array}$$

### 5.6.2 Well-definedness of operations with read and write frames

Having defined the semantics in terms of "the unique $(T, M)$ such that ...", we now show this is well defined by constructing the pair and demonstrating its existence and uniqueness.

[11]The suffix in $[\![ \; ]\!]_0$ is used to distinguish this interpretation from that defined in the next chapter.

**Theorem** For $\mathcal{S}$ defined as above we have

$$\exists!\,(T, M) \in \mathcal{S} \cdot \forall (T_i, M_i) \in \mathcal{S} \cdot T \subseteq T_i \wedge M \supseteq M_i$$

**Proof** Existence within $\mathcal{D}_o$ and uniqueness within $\mathcal{S}$ are both trivial on constructing

$$T = \bigcap_{(T_i, M_i) \in \mathcal{S}} T_i \quad \text{and} \quad M = \bigcup_{(T_i, M_i) \in \mathcal{S}} M_i$$

It remains to show "closure", that the pair constructed above is in the set $\mathcal{S}$, i.e.

$$(T, M) \in \mathcal{S}$$

To demonstrate this we show that $(T, M)$ satisfies the four properties defining $\mathcal{S}$. Each property follows easily from the assumption that all $(T_i, M_i) \in \mathcal{S}$ by various distributive properties of $\mathsf{dom}$ and $\lhd$ over $\subseteq$ and $\cup$. It is interesting to note that the proof of each of the properties relies solely on the assumption of the respective property for the $(T_i, M_i)$.

*dom*

$$\begin{array}{rcll}
\overline{T} & = & \bigcap_i T_i & \text{by def'n} \\
 & \supseteq & \bigcap_i \overline{P \wedge I} & \text{by assumption } \mathit{dom} \text{ on the } T_i \text{, and } \subseteq \text{ conjunctive} \\
 & = & \overline{P \wedge I} & \text{by idempotence of } \cap
\end{array}$$

*res*

$$\begin{array}{rcll}
\overline{P \wedge I} \lhd M & = & \overline{P \wedge I} \lhd \bigcup_i M_i & \text{by def'n} \\
 & = & \bigcup_i \overline{P \wedge I} \lhd M_i & \text{by } \lhd \text{ disjuntive} \\
 & \subseteq & \bigcup_i \overline{Q \wedge I} & \text{by } \mathit{res} \text{ on the } M_i \text{, and } \subseteq \text{ disjunctive} \\
 & = & \overline{Q \wedge I} & \text{by } \cup \text{ idempotent}
\end{array}$$

*writes*

$$\begin{array}{rcll}
M & = & \bigcup_i M_i & \text{def'n} \\
 & \supseteq & \bigcup_i \Xi_{F-W} & \text{by } \mathit{writes} \text{ on the } M_i \text{, and } \subseteq \text{ disjunctive} \\
 & = & \Xi_{F-W} & \text{by } \cup \text{ idempotent}
\end{array}$$

*readsM*

$$
\begin{aligned}
\Xi_R ; M &= \Xi_R ; \bigcup_i M_i && \text{def'n} \\
&= \bigcup_i (\Xi_R ; M_i) && ;\ \text{is disjunctive} \\
&\subseteq \bigcup_i (M_i ; \Xi_R) && \text{by } \mathit{reads} \text{ on the } M_i, \text{ and } \subseteq \text{ disjunctive} \\
&= (\bigcup_i M_i) ; \Xi_R && ;\ \text{is disjunctive} \\
&= M ; \Xi_R
\end{aligned}
$$

*readsT*

$$
\begin{aligned}
\Xi_R (\!|\, T \,|\!) &= \Xi_R (\!|\, \bigcap_i T_i \,|\!) && \text{def'n} \\
&= \bigcap_i \Xi_R (\!|\, T_i \,|\!) && (\!|\_|\!) \text{ is conjunctive} \\
&\subseteq \bigcap_i T_i && \text{by } \mathit{readsT} \text{ on the } T_i, \text{ and } \cap \text{ monotonic} \\
&= T && \text{def'n}
\end{aligned}
$$

### 5.6.3 Consistency

We conclude with a remark that the $(T, M)$ pair defined above corresponds in spirit to that in the Jones' semantics. However, as different assumptions have been made along the way concerning the treatment of the state as a single unit or a product of the spaces spanned by individual variables, and as Jones' semantics does not deal directly with the read and write frames this "consistency" can only be explained informally.

To see the correspondence, for the termination set, note that we have defined

$$T \subseteq \bigcap_{T_i \supseteq \overline{P \wedge I} \ \wedge \ \mathit{readsT}} T_i$$

which immediately gives $T = \overline{P \wedge I}$ provided $P \wedge I$ respects the read frame. For the meaning relation, we have

$$M = \bigcup_{\overline{P \wedge I} \lhd M_i \subseteq \overline{Q \wedge I} \ \wedge \ \mathit{writes} \ \wedge \ \mathit{readsM}} M_i$$

which corresponds to Jones' meaning relation with the added restriction that it is the largest such relation which satisfies the read constraint as described in Chapter 4.

## 5.7 Onwards ...

We have given a more formal treatment of the semantics of operations outlined in Chapter 4 where read and write frames can be handled independently of each other and of the operation's predicates. In passing, we noted a similarity between the definition of non-reading and that of bisimulation from process algebra. Insight gained thus lead us to a reformulation of the definition of non-reading and bisimulation.

However, this model of operations still suffers from the problem regarding the preservation of read frames by refinement described in Section 4.5. The next chapter deals with this shortcoming by enriching the model of operations to encode all read-respecting refinements into the semantics of an operation definition.

# Chapter 6

# Refinement with Read Frames

*"It seems that mathematical ideas are arranged somehow in strata..."*

## 6.1 Introduction

We saw in Chapter 4 that the usual semantic model for operations is not rich enough to ensure the preservation of read-respecting behaviour by refinement. The impoverishment arises because this model is based on the assumption that the set of refinements of an operation can be characterised by the interpretation of the most general implementation. However, this approach is not expressive enough to support the preservation by refinement of properties of non-interference which arise from read and write frames.

One approach to resolving this difficulty is to restrict the criterion for refinement so that it does indeed respect the read frame. This is the motivation for the rules for refinement presented in Chapter 3, where the definition of refinement is, in effect, parameterised by the read frame. An alternative approach is to define a richer semantic model for specifications, where operation definitions are interpreted as the set of relations on states that correspond to their valid implementations. Then it is possible to filter the putative implementations by the read constraint, thus encoding the role of the read frame as "advanced information" about implementations into the semantics of the operation specification. With this interpretation, a fixed definition of refinement, as containment of implementations, does respect the read frame.

This chapter defines such a semantics for operations and refinement. In effect, it simply moves the definition of refinement into the semantics of the individual operations, however, this adjustment ensures the preservation of non-interference by implementations and hence allows a greater degree of compositionality in development.

### 6.1.1 Examples

We begin with a demonstration of the intended interpretation by revisiting the example given in Chapters 3 and 4. Recall

$$\Sigma :: a : \mathbb{N}$$
$$\quad b : \mathbb{N}$$

$$inv\text{-}\Sigma(a, b) \quad \triangleq$$
$$a \in \{0, 1\} \wedge$$
$$b \leq a$$

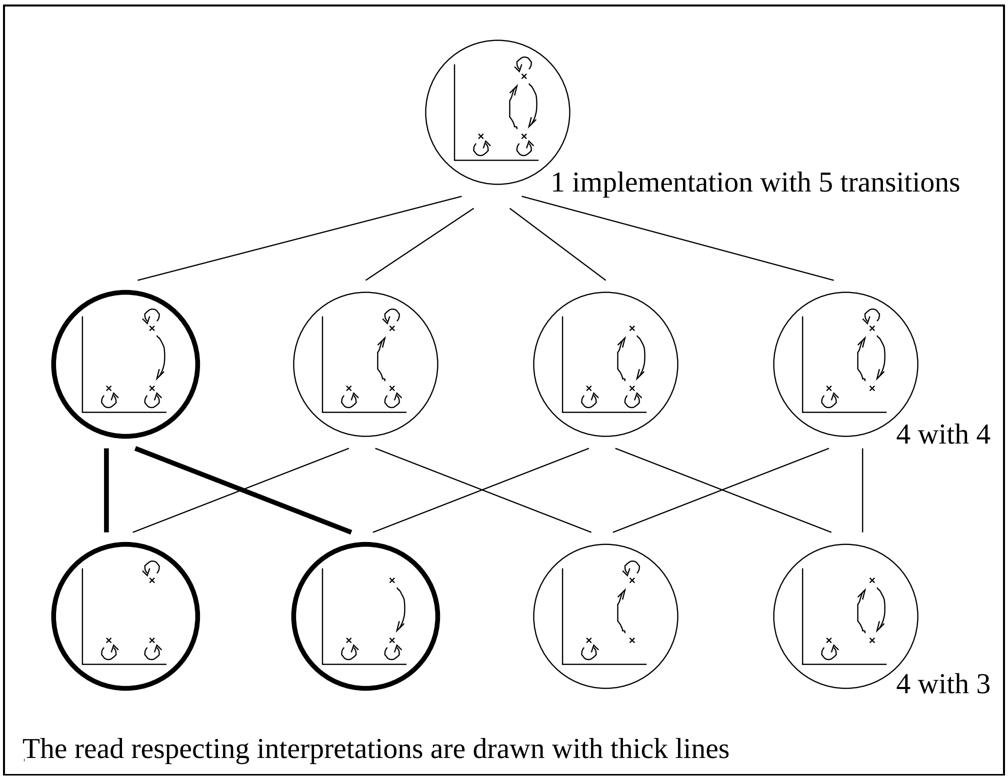
*choose_b*

ext rd $b$ : $\mathbb{N}$
 wr $b$ : $\mathbb{N}$

pre true

post true

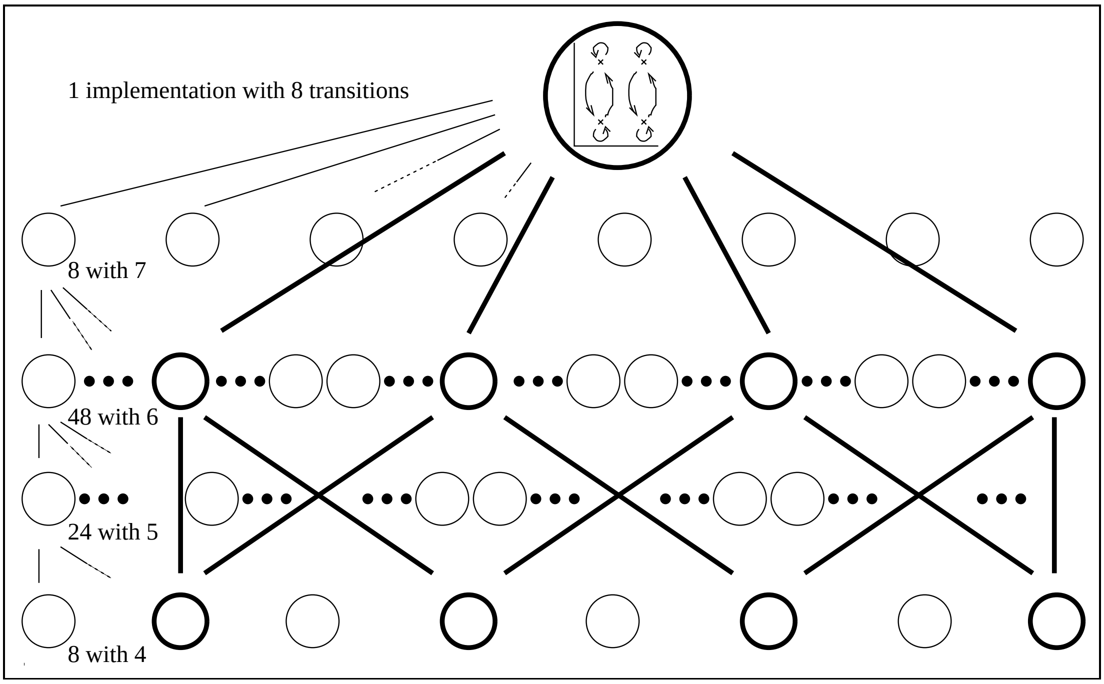
We can depict the meaning relations of the operation *choose_b* and its implementations as follows:




In the first instance, ignoring the read frame, the most general interpretation has five valid state transitions with two points of non-determinacy and is depicted at the top of the picture. There are four refinements where non-determinacy is reduced by dropping one possible transition and a further four where two are dropped.

Of these nine interpretations, three satisfy the read frame constriant. These are depicted with thick lines. In this example, the most general read respecting specification characterises, via the refinement relation, the set of all the read respecting implementations. However, this good fortune does not apply in general.

Now consider another example, similar to that above but with the clause $b \leq a$ in the invariant replaced by $b \in \{0, 1\}$. There are now four valid states and many more possible implementations of *choose_b* as depicted below




Again ignoring the read frame, there are eighty-nine different interpretations satisfying the specification of *choose_b*. In this case, the most abstract specification has eight transitions, with four points of non-determinacy, and the refinements arise by reducing the non-determinacy by removing one, two, three, or four transitions whilst maintaining the totality of the operation. (There are eight interpretations with seven transitions, forty-eight with six, twenty-four with five, and eight with four.)

In this case, 'filtering' the interpretations by the read frame constraint does not affect the most general interpretation and thus would have no effect on the standard semantics. However, all but eight of the other interpretations are removed by the read constriant. So, with this example, we cannot characterise the read-respecting implementations by a single relation and the usual definition of refinement[1].

So we see that the process of refinement is one of cutting down the set of possible behaviours as design decisions weed-out unwanted implementations. If interpreted as advanced information about code, then read restrictions give an initial sifting of the possibilities even though this may have no effect on the most abstract interpretation.

[1]The similiarity between the picture for the read-respecting interpretations in this example and the picture for all interpretations in the previous example would seem to be no more than coincidental.

## 6.2 Semantics for operation refinement

We now enrich the semantic model to bring all the possible implementations into the interpretation of an operation definition. (An implementation here does not necessarily mean an explicit operation, but could simply be a more concrete implicit operation or explicit operation with implicit "holes".)

### 6.2.1 Definitions

#### Operation semantics

Recall the semantic model of operations from Chapter 5

$$\begin{aligned}&[\![ \textit{mk-OpDef}(F, I, R, W, P, Q) ]\!]_0 \quad \triangleq \\ &\qquad \iota(T, M) \in \mathcal{S} \cdot \forall (T_i, M_i) \in \mathcal{S} \cdot T \subseteq T_i \ \wedge\ M \supseteq M_i\end{aligned}$$

where $\mathcal{S}$ is the set of all pairs satisfying the four predicates

$$\mathcal{S} = \left\{ (T_i, M_i) \colon \mathcal{D}_0 \;\middle|\; drwr_{(F,I,R,W,P,Q)}(T_i, M_i) \right\}$$

and

$$\begin{aligned} drwr_{(F,I,R,W,P,Q)}(T, M) \quad \triangleq \quad & dom_{(F,I,R,W,P,Q)}(T, M) \\ \wedge \quad & res_{(F,I,R,W,P,Q)}(T, M) \\ \wedge \quad & writes_{(F,I,R,W,P,Q)}(T, M) \\ \wedge \quad & reads_{(F,I,R,W,P,Q)}(T, M) \end{aligned}$$

In this chapter, operations are interpreted as their set of implementations and thus we take the whole set as the semantics

$$[\![ \textit{mk-OpDef}(F, I, R, W, P, Q) ]\!] \quad \triangleq \quad \mathcal{S}$$

So $[\![ \textit{mk-OpDef}(F, I, R, W, P, Q) ]\!] \colon \mathcal{D}_1$ where $\mathcal{D}_1 = \mathcal{P}\mathcal{D}_0$.

#### Refinement

With this model, refinement is simply set containment

$$Op_1 \sqsubseteq Op_2 \quad \triangleq \quad [\![ Op_1 ]\!] \supseteq [\![ Op_2 ]\!]$$

which can be expressed in terms of *drwr* as follows

$$\begin{aligned} [\![ Op_1 ]\!] \supseteq [\![ Op_2 ]\!] \quad & \Leftrightarrow \quad \forall (T, M) \in [\![ Op_2 ]\!] \cdot (T, M) \in [\![ Op_1 ]\!] \\ & \Leftrightarrow \quad \forall (T, M) \in \mathcal{D}_0 \cdot drwr_2(T, M) \ \Rightarrow\ drwr_1(T, M) \end{aligned}$$

where $Op_i = mk\text{-}OpDef(F_i, I_i, R_i, W_i, P_i, Q_i)$, for $i = 1, 2$ and

$$drwr_i(T, M) = drwr_{(F_i, I_i, R_i, W_i, P_i, Q_i)}(T, M)$$

### 6.2.2 Consistency

The semantics of Chapter 5 is thus a "retrieval" of the present model defined by

$$retr : \mathcal{D}_1 \to \mathcal{D}_0$$
$$retr(\mathcal{S}) \triangleq \left( \bigcap_{(T_i, M_i) \in \mathcal{S}} T_i , \bigcup_{(T_i, M_i) \in \mathcal{S}} M_i \right)$$

where, for all operations $Op = mk\text{-}OpDef(F, I, R, W, P, Q)$, we have (by Section 5.6.2) that $retr$ is consistent, i.e.

$$[\![Op]\!]_0 = retr([\![Op]\!])$$

Now refinement in $\mathcal{D}_1$ is a sufficient condition for refinement in $\mathcal{D}_0$ although it is not a necessary condition since the new definition of refinement prohibits putative refinements which do not respect the read frame.

**Theorem** $retr$ **preserves refinement**

$$\frac{Op_a \sqsubseteq Op_c}{Op_a \sqsubseteq_0 Op_c}$$

where $\sqsubseteq_0$ is satisfaction in the $\mathcal{D}_0$ semantics.

**Proof**

$$\begin{array}{lll}
Op_a \sqsubseteq Op_c & \triangleq & [\![Op_a]\!] \supseteq [\![Op_c]\!] \\
 & \Rightarrow & \bigcap_{(T_i,M_i)\in[\![Op_a]\!]} T_i \subseteq \bigcap_{(T_i,M_i)\in[\![Op_c]\!]} T_i \wedge \bigcup_{(T_i,M_i)\in[\![Op_a]\!]} M_i \supseteq \bigcup_{(T_i,M_i)\in[\![Op_c]\!]} M_i \\
 & \Rightarrow & \bigcap_{(T_i,M_i)\in[\![Op_a]\!]} T_i \subseteq \bigcap_{(T_i,M_i)\in[\![Op_c]\!]} T_i \wedge \\
 & & \quad \bigcup_{(T_i,M_i)\in[\![Op_a]\!]} M_i \supseteq \bigcap_{(T_i,M_i)\in[\![Op_a]\!]} T_i \lhd \bigcup_{(T_i,M_i)\in[\![Op_c]\!]} M_i \\
 & \triangleq & Op_a \sqsubseteq_0 Op_c
\end{array}$$

### 6.2.3 Miracles

Note that when no implementation is possible then $[\![Op]\!]$ is empty. This corresponds to miracle [Mor88a] since

$$retr(\{\,\}) = (\bigcap_{\{\}}, \bigcup_{\{\}}) = (\overline{\mathsf{true}}, \overline{\mathsf{false}}) = [\![\mathsf{miracle}]\!]_0$$

### 6.2.4 Soundness and Completeness

In the next section, this semantics is used as a basis for demonstrating the soundness of the proof rules from Chapter 3. Note, however, that this semantics is still not complete as, although it addresses incompleteness arising from reduction in non-determinism, it does not address incompleteness arising from non-essential read and write accesses. For example, if $y$ is not read then the semantics will not allow $x \in \{0, 1\}$ to be refined by $x := \mathsf{if}\ y = 0\ \mathsf{then}\ 0\ \mathsf{else}\ 1$, whilst the refinement $x := \mathsf{if}\ y = 0\ \mathsf{then}\ 0\ \mathsf{else}\ 0$ is permitted since this is equivalent to $x := 0$. Thus the semantics does admit refinements which *could* arise from code which syntactically breaks the read and write frame provided that they could also arise from read-write-respecting code. On the other hand, the proof rules will not allow either alternative since they prohibit the development of code which syntactically breaks the read frame even if such accesses have no effect on the resulting behaviour.

## 6.3 Explicit operation constructors

Having defined a richer semantics for implicit operations and refinement, we now define a semantics for the explicit operation constructors which mirrors that of [Jon87]. We then show that each construct maintains the consistency of *retr*, is monotonic in the refinement ordering and maintains feasibility. Finally, we show that proof rules given in Chapter 3 are sound with respect to this semantics.

We first discuss the approach to be taken in a general setting.

### 6.3.1 General approach: Definition and consistency

For each form of statement construction, say for a binary construct, $C$, let

$$Op_C = C[Op_1, Op_2]$$

for some $Op_i = mk\text{-}OpDef(F_i, I_i, R_i, W_i, P_i, Q_i)$, $i = 1, 2$, and let

$$(T_i, M_i) = [\![Op_i]\!]_0$$

Now, where [Jon87] defines

$$[\![Op_C]\!]_0 = expr(T_1, M_1, T_2, M_2)$$

for some construction $expr$, we define the semantics of $C$ by

$$\begin{aligned} [\![Op_C]\!] &\triangleq \{expr(T_1, M_1, T_2, M_2) \mid (T_1, M_1) \in [\![Op_1]\!], (T_2, M_2) \in [\![Op_2]\!]\} \\ &= \left\{ (T, M) : \mathcal{D}_0 \;\middle|\; \begin{array}{c} \exists (T_1, M_1) \in [\![Op_1]\!], (T_2, M_2) \in [\![Op_2]\!] \cdot \\ (T, M) = expr(T_1, M_1, T_2, M_2) \end{array} \right\} \end{aligned}$$

So to demonstrate consistency for each statement form, we must show that this construction maintains the consistency of $retr$. That is

$$retr([\![Op_C]\!]) = expr(retr([\![Op_1]\!]), retr([\![Op_2]\!]))$$

### 6.3.2 General approach: Feasibility

Define feasibility as

$$feasible(Op) \quad \triangleq \quad \exists (T, M) \in [\![Op]\!] \cdot T \subseteq \mathsf{dom}\ M$$

so that an operation is feasible if and only if it has a feasible implementation.

Then, for the same operation construction as above, we need to show

$$\frac{\begin{array}{c} feasible(Op_1) \\ feasible(Op_2) \end{array}}{feasible(Op_C)}$$

i.e.

$$\frac{\begin{array}{c} \exists (T_1, M_1) \in [\![Op_1]\!] \cdot T_1 \subseteq \mathsf{dom}\ M_1 \\ \exists (T_2, M_2) \in [\![Op_2]\!] \cdot T_2 \subseteq \mathsf{dom}\ M_2 \end{array}}{\exists (T, M) \in [\![Op_C]\!] \cdot T \subseteq \mathsf{dom}\ M}$$

We show this by constructing $(T, M) = expr(T_1, M_1, T_2, M_2)$ as in the definition of the semantics of $C$ and then proving

$$\frac{\begin{array}{c} T_1 \subseteq \mathsf{dom}\ M_1 \\ T_2 \subseteq \mathsf{dom}\ M_2 \\ (T, M) = expr(T_1, M_1, T_2, M_2) \end{array}}{T \subseteq \mathsf{dom}\ M}$$

which is exactly the same as the condition from [Jon87].

### 6.3.3 General approach: Monotonicity

In this setting monotonicity is completely trivial. The requirement is

$$\frac{\begin{array}{c} Op_1 \sqsubseteq Op_1' \\ Op_2 \sqsubseteq Op_2' \end{array}}{C[Op_1, Op_2] \sqsubseteq C[Op_1', Op_2']}$$

which follows immediately by construction, since

$$[\![C[Op_1, Op_2]]\!] = \left\{ expr(T_1, M_1, T_2, M_2) \;\middle|\; (T_i, M_i) \in [\![Op_i]\!] \right\}$$

and set comprehension over a smaller set clearly leads to a smaller set of constructions.

### 6.3.4 General approach: Soundness

Suppose that we have a proof rule for the introduction of the construction $C$ of the form

$$\frac{hyps}{Op_A \sqsubseteq C[Op_1, Op_2]}$$

To show soundness of this proof rule we need to show

$$\frac{hyps}{[\![Op_A]\!] \supseteq [\![C[Op_1, Op_2]]\!]}$$

that is

$$\frac{\begin{array}{c} hyps \\ (T, M) \in [\![C[Op_1, Op_2]]\!] \end{array}}{(T, M) \in [\![Op_A]\!]}$$

Now, recall that we have

$$\begin{array}{rcl} (T, M) \in [\![C[Op_1, Op_2]]\!] & \Leftrightarrow & \exists (T_1, M_1) \in [\![Op_1]\!], (T_2, M_2) \in [\![Op_2]\!] \cdot \\ & & \quad (T, M) = expr(T_1, M_1, T_2, M_2) \end{array}$$

and, for any $(T, M) \colon \mathcal{D}_0$, we also have

$$(T, M) \in [\![Op_i]\!] \;\Leftrightarrow\; drwr_i(T, M)$$

so the above becomes

$$\frac{\begin{array}{c} hyps \\ drwr_1(T_1, M_1) \\ drwr_2(T_2, M_2) \\ (T, M) = expr(T_1, M_1, T_2, M_2) \end{array}}{drwr_A(T, M)}$$

To show this we will, in fact, show each of the four predicates in the conclusion separately. Interestingly, as in the proof of closure of $\mathcal{S}$ in Chapter 5, on the whole, the proof of each property only requires the assumption of the respective property on the $(T_i, M_i)$. The one exception to this arises in the proof of soundness of the rule for contracting the read frame (Section 6.8.4) and is discussed in more detail in Section 7.2.

## 6.4 Sequential composition of operations

We begin with sequential composition which is straightforward as the read frame plays no significant part in its semantics.

### 6.4.1 Definition

We define the semantics of sequential composition corresponding to that in [Jon87].

$$[\![Op_1 ; Op_2]\!] \triangleq \left\{ (T, M) : \mathcal{D}_o \;\middle|\; \begin{array}{l} \exists (T_1, M_1) \in [\![Op_1]\!], (T_2, M_2) \in [\![Op_2]\!] \cdot \\ \quad M = M_1 ; M_2 \;\wedge \\ \quad T = T_1 - \mathsf{dom}\,(M_1 \mathrel{\triangleright} T_2) \end{array} \right\}$$

As in [Jon87], the need for"angelic non-determinism" is avoided by the constraint on the termination set.

### 6.4.2 Consistency

Let $[\![Op_1]\!] = \{(T_i, M_i) \mid i \in S_1\}$ and $[\![Op_2]\!] = \{(T_j, M_j) \mid j \in S_2\}$

Following the general approach for proving consistency from Section 6.3.1, we are required to prove

$$\bigcup_{i \in S_1, j \in S_2} M_i ; M_j \;=\; \bigcup_{i \in S_1} M_i \;;\; \bigcup_{j \in S_2} M_j$$

and

$$\bigcap_{i \in S_1, j \in S_2} (T_i - \mathsf{dom}\,(M_i \mathrel{\triangleright} T_j)) \;=\; \bigcap_{i \in S_1} T_i \;-\mathsf{dom}\,(\bigcup_{i \in S_1} M_i \;\triangleright\; \bigcap_{j \in S_2} T_j\;)$$

#### Proof

The first condition follows easily as sequencing is disjunctive, for the second we see

$$
\begin{array}{lll}
\multicolumn{2}{l}{\bigcap_{i,j}(T_i - \mathsf{dom}\,(M_i \mathrel{\ntriangleright} T_j))} & \\
\quad = & (\bigcap_i T_i) - \bigcup_{i,j} \mathsf{dom}\,(M_i \mathrel{\ntriangleright} T_j) & \text{by } \bigcap(S - T) = (\bigcap S) - (\bigcup T) \\
\quad = & (\bigcap_i T_i) - \mathsf{dom}\,\bigcup_{i,j}(M_i \mathrel{\ntriangleright} T_j) & \text{by domain disjunctive} \\
\quad = & \bigcap_i (T_i - \mathsf{dom}\,(\bigcup_i M_i \mathrel{\ntriangleright} \bigcap_i T_j) & \text{by } \bigcup(R \mathrel{\ntriangleright} S) = (\bigcup R) - (\bigcap S)
\end{array}
$$

### 6.4.3 Feasibility

Now, following the general approach for proving feasibility in Section 6.3.2 we need to show

$$
\frac{\begin{array}{c} T_1 \subseteq \mathsf{dom}\, M_1 \\ T_2 \subseteq \mathsf{dom}\, M_2 \\ T = T_1 - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) \\ M = M_1 ; M_2 \end{array}}{T \subseteq \mathsf{dom}\, M}
$$

which can be demonstrated as follows

**Proof** *Feasibility*

$$
\begin{array}{rcll}
T & = & T_1 - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) & \text{hypothesis (3)} \\
 & \subseteq & \mathsf{dom}\, M_1 - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) & \text{hypothesis (1) } T_1 \subseteq \mathsf{dom}\, M_1 \\
 & = & \mathsf{dom}\,((M_1 \mathrel{\triangleright} T_2) \cup (M_1 \mathrel{\ntriangleright} T_2)) - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) & \\
 & & & \text{by } R = (R \mathrel{\triangleright} S) \cup (R \mathrel{\ntriangleright} S) \\
 & = & (\mathsf{dom}\, M_1 \mathrel{\triangleright} T_2) \cup (\mathsf{dom}\, M_1 \mathrel{\ntriangleright} T_2) - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) & \textit{dom} \text{ distr.} \\
 & \subseteq & \mathsf{dom}\, M_1 \mathrel{\triangleright} T_2 & S \cup T - T \subseteq S \\
 & \subseteq & \mathsf{dom}\,(M_1 \mathrel{\triangleright} \mathsf{dom}\, M_2) & \text{hypothesis (2) } T_2 \subseteq \mathsf{dom}\, M_2, \text{ and } \triangleright \text{ mono.} \\
 & = & \mathsf{dom}\,(M_1 ; M_2) & \text{lemma: } \mathsf{dom}\,(R_1 ; R_2) = \mathsf{dom}\,(R_1 \mathrel{\triangleright} \mathsf{dom}\, R_2) \\
 & = & \mathsf{dom}\, M & \text{hypothesis (4)}
\end{array}
$$

### 6.4.4 Proof rule

The rule for sequential composition given in Chapter 3 is

$$
\frac{\begin{array}{c} \overleftarrow{P \wedge I} \wedge Q_1 \wedge I \;\Rightarrow\; P_2 \\ Q_1 ; Q_2 \;\Rightarrow\; Q \end{array}}{(F, I, R, W, P, Q) \sqsubseteq (F, I, R, W, P, Q_1);(F, I, R, W, P_2, Q_2)}
$$

### 6.4.5 Soundness

Following the approach described in Section 6.3.4, to demonstrate soundness we are required to prove

$$\begin{array}{c}
\overleftarrow{P \wedge I} \wedge Q_1 \wedge I \;\Rightarrow\; P_2 \\
Q_1 ; Q_2 \;\Rightarrow\; Q \\
T = T_1 - \mathsf{dom}\,(M_1 \mathbin{\rhd\!\!\!-} T_2) \\
M = M_1 ; M_2 \\
drwr_{(F,I,R,W,P,Q_1)}(T_1, M_1) \\
drwr_{(F,I,R,W,P_2,Q_2)}(T_2, M_2) \\
\hline
drwr_{(F,I,R,W,P,Q)}(T, M)
\end{array}$$

**Proof** *Sequential Composition*

We show each of the four properties: *res*, *dom*, *writes*, and *reads* for $(T, M)$ separately. The proofs of the first two follow similar lines to the proofs in [Jon87], the third is trivial, and so *readsT* and *readsM* are the only really new thing to show.

**dom** $\overline{P \wedge I} \subseteq T$

**Proof** *dom*

Note that

$$T = T_1 - \mathsf{dom}\,(M_1 \mathbin{\rhd\!\!\!-} T_2)$$

and since we can assume

$$\overline{P \wedge I} \subseteq T_1$$

it suffices to show

$$\overline{P \wedge I} \cap \mathsf{dom}\,(M_1 \mathbin{\rhd\!\!\!-} T_2) = \{\,\}$$

Now

$$\begin{array}{rcll}
\mathsf{rng}\,(\overline{P \wedge I} \lhd M_1) & = & \mathsf{rng}\,(\overline{P \wedge I} \lhd (\overline{P \wedge I} \lhd M_1)) & \text{by } S \lhd (S \lhd R) = S \lhd R \\
& \subseteq & \mathsf{rng}\,(\overline{P} ; \overline{I} ; \overline{Q_1} ; \overline{I}) & \text{hyp } \overline{P \wedge I} \lhd M_1 \subseteq \overline{Q \wedge I} \\
& \subseteq & \mathsf{rng}\,(\overline{P} ; \overline{I} ; \overline{Q_1} ; \overline{I} ; \overline{P_2}) & \text{hyp } \overleftarrow{P \wedge I} \wedge Q_2 \wedge I \;\Rightarrow\; P_2 \\
& \subseteq & \overline{I} ; \overline{P_2} & \mathsf{rng}\,(R ; S) \subseteq S \\
& \subseteq & T_2 & \text{hyp } \overline{P_2 \wedge I} \subseteq T_2
\end{array}$$

so $\overline{P \wedge I} \lhd M_1 \nrhd T_2 = \{\}$ by the above

so $\mathsf{dom}\,(\overline{P \wedge I} \lhd M_1 \nrhd T_2) = \{\}$ since $\mathsf{dom}\,\{\} = \{\}$

so $\overline{P \wedge I} \cap \mathsf{dom}\,(M_1 \nrhd T_2) = \{\}$ since $dom(S \lhd R) = S \cap domR$

as required.

**res** $\overline{P \wedge I} \lhd M \subseteq \overline{Q \wedge I}$

**Proof** *res*

$$
\begin{array}{rcll}
\overline{P \wedge I} \lhd M & = & \overline{P \wedge I} ; M & \text{defn of } \lhd \\
& = & \overline{P} ; \overline{I} ; M_1 ; M_2 & \text{defn of } M\text{, defn of } ; \text{ on predicates} \\
& \subseteq & \overline{P} ; \overline{I} ; \overline{Q_1} ; \overline{I} ; M_2 & \text{hypothesis } \overline{P \wedge I} \lhd M_1 \subseteq \overline{Q_1 \wedge I} \\
& \subseteq & \overline{P} ; \overline{I} ; \overline{Q_1} ; \overline{I} ; \overline{P_2} ; M_2 & \text{hypothesis (1), } \overleftarrow{P \wedge I} \wedge Q_1 \wedge I \;\Rightarrow\; P_2 \\
& \subseteq & \overline{P} ; \overline{I} ; \overline{Q_1} ; Q_2 ; \overline{I} & \text{hypothesis } \overline{I \wedge P_2} \lhd M_2 \subseteq \overline{Q_2 \wedge I} \\
& \subseteq & \overline{Q_1} ; \overline{Q_2} ; \overline{I} & S ; R \subseteq R \\
& \subseteq & \overline{Q} ; \overline{I} & \text{hypothesis (2), } Q_1 ; Q_2 \;\Rightarrow\; Q \\
& = & \overline{Q \wedge I} & \text{defn of } ;
\end{array}
$$

**writes** $M \subseteq \Xi_{F-W}$

**Proof** *writes*

$$
\begin{array}{rcll}
M & = & M_1 ; M_2 & \text{defn} \\
& \subseteq & \Xi_{F-W} ; \Xi_{F-W} & \text{hypotheses } M_i \subseteq \Xi_{F-W} \text{ and } ; \text{ monotone} \\
& = & \Xi_{F-W} & \Xi_S \text{ idempotent}
\end{array}
$$

**readsM** $\Xi_R ; M \subseteq M ; \Xi_R$

**Proof** *readsM*

$$
\begin{array}{rcll}
\Xi_R ; M & = & \Xi_R ; M_1 ; M_2 & \text{defn} \\
& \subseteq & M_1 ; \Xi_R ; M_2 & \text{hypothesis } \Xi_R ; M_1 \subseteq M_1 ; \Xi_R\text{, and } ; \text{ monotonic} \\
& \subseteq & M_1 ; M_2 ; \Xi_R & \text{hypothesis } \Xi_R ; M_2 \subseteq M_2 ; \Xi_R\text{, and } ; \text{ monotonic} \\
& = & M ; \Xi_R & \text{defn}
\end{array}
$$

**readsT** $\Xi_R (\mid T \mid) \subseteq T$

**Proof** *readsT*

Let $\sigma_2 \in \Xi_R (\mid T \mid)$ be given. Then

$$\exists \sigma_1 \cdot \sigma_1 \Xi_R \sigma_2 \wedge \sigma_1 \in T$$

and so

$$\exists \sigma_1 \cdot \sigma_1 \Xi_R \sigma_2 \wedge \sigma_1 \in T_1 \wedge M_1 (\mid \{\sigma_1\} \mid) \subseteq T_2$$

since $\sigma \in T_1 - \mathsf{dom}\,(M_1 \not\triangleright T_2) \;\Leftrightarrow\; \sigma \in T_1 \wedge M_1 (\mid \sigma \mid) \subseteq T_2$

Hence

$$\sigma_2 \in \Xi_R (\mid T_1 \mid) \subseteq T_1$$

and

$$\begin{array}{rcll} M_1 (\mid \{\sigma_2\} \mid) & \subseteq & M_1 (\mid \Xi_R (\mid \{\sigma_1\} \mid) \mid) & \\ & = & (\Xi_R ; M_1) (\mid \{\sigma_1\} \mid) & \\ & \subseteq & (M_1 ; \Xi_R) (\mid \{\sigma_1\} \mid) & \text{hypothesis, } readsM(R, M) \\ & = & \Xi_R (\mid M_1 (\mid \{\sigma_1\} \mid) \mid) & \\ & \subseteq & \Xi_R (\mid T_2 \mid) & \\ & \subseteq & T_2 & \end{array}$$

so $\sigma_2 \in T_1 - \mathsf{dom}\,(M_1 \not\triangleright T_2) = T$ as required.

## 6.5 Conditional composition of operations

### 6.5.1 Definition

The definition of the semantics of a conditional statement also follows from the definition in [Jon87].

$$[\![ \mathsf{if}\ B\ \mathsf{then}\ Op_1\ \mathsf{else}\ Op_2 ]\!] \;\triangleq$$

$$\left\{ (T, M) : \mathcal{D}_o \;\middle|\; \begin{array}{l} \exists (T_1, M_1) \in [\![Op_1]\!], (T_2, M_2) \in [\![Op_2]\!] \cdot \\ \quad M = \overline{B} \lhd M_1 \cup \overline{\neg B} \lhd M_2 \wedge \\ \quad T = \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 \end{array} \right\}$$

### 6.5.2 Consistency

Again let $[\![Op_1]\!] = \{(T_i, M_i) \mid i \in S_1\}$ and $[\![Op_2]\!] = \{(T_j, M_j) \mid j \in S_2\}$

We are required to prove

$$\bigcup_{i \in S_1, j \in S_2} (\overline{B} \lhd M_i \ \cup \ \overline{\neg B} \lhd M_j) \ = (\overline{B} \lhd \bigcup_{i \in S_1} M_i \ ) \ \cup \ (\overline{\neg B} \lhd \bigcup_{j \in S_2} M_j \ )$$

and

$$\bigcap_{i \in S_1, j \in S_2} (\overline{B} \cap T_i \ \cup \ \overline{\neg B} \cap T_j) \ = (\overline{B} \cap \bigcap_{i \in S_1} T_i \ ) \ \cup \ (\overline{\neg B} \cap \bigcap_{j \in S_2} T_j \ )$$

**Proof**

The proof follows by disjunctive properties of $\lhd$ and $\bigcap$.

$$\begin{aligned}
\bigcup_{i,j} (\overline{B} \lhd M_i \ \cup \ \overline{\neg B} \lhd M_j) &= (\bigcup_i \overline{B} \lhd M_i) \ \cup \ (\bigcup_j \overline{\neg B} \lhd M_j) \\
&= (\overline{B} \lhd \bigcup_i M_i) \ \cup \ (\overline{\neg B} \lhd \bigcup_j M_j)
\end{aligned}$$

$$\begin{aligned}
\bigcap_{i,j} (\overline{B} \cap T_i \ \cup \ \overline{\neg B} \cap T_j) &= (\bigcap_i \overline{B} \cap T_i) \ \cup \ (\bigcap_j \overline{\neg B} \cap T_j) \\
&= (\overline{B} \cap \bigcap_i T_i) \ \cup \ (\overline{\neg B} \cap \bigcap_j T_j)
\end{aligned}$$

### 6.5.3 Feasibility

$$\frac{\begin{array}{c} T_1 \subseteq \mathsf{dom}\, M_1 \\ T_2 \subseteq \mathsf{dom}\, M_2 \\ T = \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 \\ M = \overline{B} \lhd M_1 \cup \overline{\neg B} \lhd M_2 \end{array}}{T \subseteq \mathsf{dom}\, M}$$

**Proof** *Feasibility*

$$\begin{array}{lcll}
T & = & \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 & \text{hypothesis} \\
 & \subseteq & \overline{B} \cap \mathsf{dom}\, M_1 \cup \overline{\neg B} \cap \mathsf{dom}\, M_2 & \text{hypothesis, } \cup \text{ monotonic} \\
 & = & \mathsf{dom}\,(\overline{B} \lhd M_1) \cup \mathsf{dom}\,(\overline{\neg B} \lhd M_2) & S \cap \mathsf{dom}\, R = \mathsf{dom}\, S \lhd R \\
 & = & \mathsf{dom}\,(\overline{B} \lhd M_1 \cup \overline{\neg B} \lhd M_2) & \mathsf{dom}\,(R_1 \cup R_2) = \mathsf{dom}\, R_1 \cup \mathsf{dom}\, R_2 \\
 & = & \mathsf{dom}\, M & \text{defn}
\end{array}$$

### 6.5.4 Proof rule

The rule for conditional composition is

$$\frac{reads(R, B)}{(F, I, R, W, P, Q) \sqsubseteq \begin{array}{l}\mathsf{if}\ B\\ \mathsf{then}\ (F, I, R, W, P \wedge B, Q)\\ \mathsf{else}\ (F, I, R, W, P \wedge \neg\, B, Q)\end{array}}$$

where the hypothesis[2], $reads(R, B)$, is an extra constraint over Jones' rule, and is defined on single state predicates as in $readsT$

$$reads(R, P) \quad \triangleq \quad \Xi_R; \overline{P} \subseteq \overline{P}; \Xi_R$$

### 6.5.5 Soundness

Required to prove

$$\frac{\begin{array}{c} reads(R, B)\\ P \;\Rightarrow\; B \vee \neg\, B\\ T = \overline{B} \cap T_1 \cup \overline{\neg\, B} \cap T_2\\ M = \overline{B} \lhd M_1 \cup \overline{\neg\, B} \lhd M_2\\ drwr_{(F,I,R,W,P \wedge B,Q)}(T_1, M_1)\\ drwr_{(F,I,R,W,P \wedge \neg\, B,Q)}(T_2, M_2)\end{array}}{drwr_{(F,I,R,W,P,Q)}(T, M)}$$

**Proof** *Conditional Composition*

We show the properties: $dom$, $res$, $writes$, $readsM$ and $readsT$.

**dom** $\overline{P \wedge I} \subseteq T$

**Proof** *dom*

[2]The rule give in Chapter 3 has the syntactic condition $B \colon Expr(R)$ which is sufficient for the semantic condition $reads(R, B)$ given here.

$$
\begin{array}{rcll}
\overline{P \wedge I} & = & \overline{P \wedge (B \vee \neg B) \wedge I} & \text{hypothesis } P \Rightarrow B \vee \neg B \\
& = & \overline{P \wedge B \wedge I} \cup \overline{P \wedge \neg B \wedge I} & \text{prop} \\
& = & \overline{B \wedge P \wedge B \wedge I} \cup \overline{\neg B \wedge P \wedge \neg B \wedge I} & \text{prop} \\
& = & \overline{B} \cap \overline{P \wedge B \wedge I} \cup \overline{B} \cap \overline{P \wedge \neg B \wedge I} & \text{set theory} \\
& \subseteq & \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 & \text{assumptions, } dom \text{ for } T_i \\
& = & T & \text{defn}
\end{array}
$$

**res** $\overline{P \wedge I} \lhd M \subseteq \overline{Q \wedge I}$

**Proof** *res*

$$
\begin{array}{rcll}
\overline{P \wedge I} \lhd M & = & \overline{P \wedge I};M & \text{defn of } \lhd \\
& = & \overline{P};\overline{I};(\overline{B};M_1 \cup \overline{\neg B};M_2) & \text{defn of } M \\
& = & \overline{P};\overline{I};\overline{B};M_1 \cup \overline{P};\overline{I};\overline{\neg B};M_2) & \cup \text{ distributes over } ; \\
& = & (\overline{P};\overline{I};\overline{B}) \lhd M_1 \cup (\overline{P};\overline{I};\overline{\neg B}) \lhd M_2) & \text{defn} \\
& \subseteq & \overline{Q};\overline{I} \cup \overline{Q};\overline{I} & \text{hypothesis, } \cup \text{ monotonic in } \subseteq \\
& = & \overline{Q};\overline{I} & \cup \text{ idempotent}
\end{array}
$$

**writes** $M \subseteq \Xi_{F-W}$

**Proof** *writes*

$$
\begin{array}{rcll}
M & = & \overline{B};M_1 \cup \overline{\neg B};M_2 & \text{defn} \\
& \subseteq & M_1 \cup M_2 & S \lhd R \subseteq R \\
& \subseteq & \Xi_{F-W} \cup \Xi_{F-W} & \text{hypothesis} \\
& = & \Xi_{F-W} & \text{hypothesis}
\end{array}
$$

**readsM** $\Xi_R;M \subseteq M;\Xi_R$

**Proof** *readsM*

$$
\begin{array}{lll}
\Xi_R;M & = \Xi_R;(\overline{B} \lhd M_1 \cup \overline{\neg B} \lhd M_2) & \text{defn} \\
& = (\Xi_R;\overline{B};M_1) \cup (\Xi_R;\overline{\neg B};M_2) & ;\ \text{is disjunctive, } S \lhd R = S;R \\
& = (\overline{B};\Xi_R;M_1) \cup (\overline{\neg B};\Xi_R;M_2) & \\
& & \text{by lemma } \Xi_R;\overline{B} = \overline{B};\Xi_R \text{ and } \Xi_R;\overline{\neg B} = \overline{\neg B};\Xi_R \\
& \subseteq (\overline{B};M_1;\Xi_R) \cup (\overline{\neg B};M_2;\Xi_R) & \\
& & \text{hypothesis, ; disjunctive and monotonic in } \subseteq \\
& = (\overline{B} \lhd M_1;\Xi_R) \cup (\overline{\neg B} \lhd M_2;\Xi_R) & S \lhd R = S;R \\
& = (\overline{B};M_1 \cup \overline{\neg B};M_2);\Xi_R & ;\ \text{is disjunctive} \\
& = M;\Xi_R & \text{defn}
\end{array}
$$

**readsT** $\Xi_R (\!| T |\!) \subseteq T$

**Proof** *readsT*

$$
\begin{array}{lll}
\Xi_R (\!| T |\!) & = \Xi_R (\!| \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 |\!) & \text{defn} \\
& = \Xi_R (\!| \overline{B} |\!) \cap \Xi_R (\!| T_1 |\!) \cup \Xi_R (\!| \overline{\neg B} |\!) \cap \Xi_R (\!| T_2 |\!) & \\
& & (\!| \_ |\!) \text{ is disjunctive and conjunctive} \\
& \subseteq \overline{B} \cap T_1 \cup \overline{\neg B} \cap T_2 & \\
& & \mathit{reads}(T, R),\ \mathit{reads}(\overline{B}, R),\ \mathit{reads}(\overline{\neg B}, R),\ \cup \text{ and } \cap \text{ monotonic} \\
& = T & \text{defn}
\end{array}
$$

# 6.6 Assignment

## 6.6.1 Definition

We define the semantics of assignment in the obvious way

$$
[\![ x := e ]\!] \;\triangleq\; \left\{ (T, M) : \mathcal{D}_o \;\middle|\; \begin{array}{l} T = \mathsf{true} \wedge \\ M = \left\{ (\overleftarrow{\sigma}, \sigma) \;\middle|\; \sigma = \overleftarrow{\sigma} \left[ e(\overleftarrow{x})/x \right] \right\} \end{array} \right\}
$$

for simplicity, we assume that $e$ is total and deterministic[3].

Note that

$$
M = \left\{ (\overleftarrow{\sigma}, \sigma) \;\middle|\; \overleftarrow{\sigma}\ \Xi_{F\text{-}\{x\}} \sigma \ \wedge\ \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma}) \right\}
$$

[3]However, if $e$ were partial then its domain of definedness could be given in $T$, and if it were under-specified then the different interpretations could be given in the $M_i$. If looseness is to be interpreted as non-determinism then the non-deterministic relation should be used as well as any deterministic sub-relations. If it is underspecification, then only the deterministic relations should be used. Thus this interpretation can distinguish between the two interpretations of looseness.

### 6.6.2 Consistency

Consistency is trivial since $[\![x := e]\!]$ is a singleton set.

### 6.6.3 Feasibility

$$\overline{\mathsf{true}} \subseteq \mathsf{dom}\ M$$

**Proof** *Feasibility*

Feasibility holds trivially provided $M$ is total, which is the case provided $e$ is total.

### 6.6.4 Proof rule

The proof rule for assignment in Chapter 3 is

$$\frac{x \in W,\ e : Exp(R),\ Id_{F-\{x\}} \wedge I \wedge P \ \Rightarrow\ (Q \wedge I)\left[e/x, x/\overleftarrow{x}\right]}{(F, I, R, W, P, Q) \sqsubseteq x := e}$$

### 6.6.5 Soundness

As this is a primitive construct we do not have assumptions about the component statements, rather we assume that $T$ and $M$ are given directly according to the semantics. Thus we need to show

$$\frac{\begin{array}{c} x \in W \\ e : Exp(R) \\ Id_{F-\{x\}} \wedge I \wedge P \ \Rightarrow\ (Q \wedge I)\left[e/x, x/\overleftarrow{x}\right] \\ T = \overline{\mathsf{true}} \\ M = \left\{(\overleftarrow{\sigma}, \sigma) \,\middle|\, \overleftarrow{\sigma}\ \Xi_{F\text{-}\{x\}}\sigma \wedge \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma})\right\} \end{array}}{drwr_{(F,I,R,W,P,Q)}(T, M)}$$

Note that the first two hypotheses are really syntactic conditions that perhaps would be better expressed as side conditions.

**dom** $\overline{P \wedge I} \subseteq T$

**Proof** *dom*

Trivial.

**res** $\overline{P \wedge I} \lhd M \subseteq \overline{Q \wedge I}$

**Proof** **res**

$$
\begin{array}{ll}
\overline{P \wedge I} \lhd M & \\
= \left\{ (\overleftarrow{\sigma}, \sigma) \,\middle|\, \overleftarrow{P} \wedge \overleftarrow{I} \wedge \overleftarrow{\sigma} \ \Xi_{F\text{-}\{x\}} \ \sigma \wedge \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma}) \right\} & \text{defn of } M \\
= \left\{ (\overleftarrow{\sigma}, \sigma) \,\middle|\, (P \wedge I \wedge \sigma \ \Xi_{F\text{-}\{x\}} \ \overleftarrow{\sigma}) \left[\overleftarrow{\sigma}/\sigma, \sigma/\overleftarrow{\sigma}\right] \wedge \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma}) \right\} & \\
& \text{renaming } \sigma \text{ and } \overleftarrow{\sigma} \\
\subseteq \left\{ (\overleftarrow{\sigma}, \sigma) \,\middle|\, (Q \wedge I) \left[e(\sigma)/x, x/\overleftarrow{x}\right] \left[\overleftarrow{\sigma}/\sigma, \sigma/\overleftarrow{\sigma}\right] \wedge \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma}) \right\} & \\
& \text{by hypothesis } Id_{F-\{x\}} \wedge I \wedge P \ \Rightarrow \ (Q \wedge I) \left[e/x, x/\overleftarrow{x}\right] \\
= \left\{ (\overleftarrow{\sigma}, \sigma) \,\middle|\, (Q \wedge I) \left[e(\overleftarrow{\sigma}/x, \overleftarrow{\sigma}\big|_{F\text{-}\{x\}}/\sigma\big|_{F\text{-}\{x\}}\right] \wedge \sigma\big|_{\{x\}} = e(\overleftarrow{\sigma}) \right\} & \\
& \text{doing the substitution} \\
= \left\{ (\overleftarrow{\sigma}, \sigma) \,\middle|\, (Q \wedge I) \right\} & \text{by replacement of equals} \\
= \overline{Q \wedge I} &
\end{array}
$$

**writes** $M \subseteq \Xi_{F\text{-}W}$

**Proof** *writes*

This follows easily on noting that

$$\forall (\overleftarrow{\sigma}, \sigma) \in M \cdot \overleftarrow{\sigma} \ \Xi_{F\text{-}\{x\}} \ \sigma$$

and

$$\{x\} \subseteq W$$

**readsM**

**Proof** *readsM*

let $\overleftarrow{\sigma_1}(\Xi_R ; M)\sigma_2$, now

$$
\begin{array}{lll}
\overleftarrow{\sigma_1}(\Xi_R ; M)\sigma_2 & \Leftrightarrow & \exists \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1} \ \Xi_R \ \overleftarrow{\sigma_2} \ M \ \sigma_2 \\
& \Leftrightarrow & \exists \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1} \ \Xi_R \ \overleftarrow{\sigma_2} \ \Xi_{F-\{x\}} \ \sigma_2 \wedge \sigma_2\big|_{\{x\}} = e(\overleftarrow{\sigma_2})
\end{array}
$$

Now choose $\sigma_1$ s.t. $\overleftarrow{\sigma_1}\ M\ \sigma_1$. (The state $\sigma_1$ exists since assignment is total.) ie.

$$\sigma_1\Big|_{\{x\}} = e(\overleftarrow{\sigma_1}) \quad \wedge$$

$$\sigma_1\Big|_{F-\{x\}} = \overleftarrow{\sigma_1}\Big|_{F-\{x\}}$$

But now $\sigma_1\Big|_{R-\{x\}} = \sigma_2\Big|_{R-\{x\}}$ since all four states are equal on $R - \{x\}$

and $\sigma_1\Big|_{\{x\}} = e(\overleftarrow{\sigma_1}) = e(\overleftarrow{\sigma_2}) = \sigma_2\Big|_{\{x\}}$ since $\overleftarrow{\sigma_1}\ \Xi_R\ \overleftarrow{\sigma_2}$ and $e\colon Expr(R)$.

So $\sigma_1\ \Xi_{R-\{x\}}\ \sigma_2$ and hence $\overleftarrow{\sigma_1}(M;\Xi_R)\sigma_2$ as required.

**readsT**

This is trivial since $T = \Sigma$ and $\Xi_R\,(\!|\,\Sigma\,|\!) = \Sigma$.

## 6.7 Iteration

The incorporation of read and write frames into the treatment of iteration is similar to their incorporation into alternation and sequencing.

### 6.7.1 Definition

The definition of the semantics for while follows similarly from the definition in [Jon87].

$$[\![\mathsf{while}\ B\ \mathsf{do}\ Op\ \mathsf{end}]\!] \triangleq$$
$$\left\{ (T, M)\colon \mathcal{D}_o \;\middle|\; \begin{array}{l} \exists (T_1, M_1) \in [\![Op]\!]\cdot \\ \quad M = \mathit{fix}(\lambda r\cdot(\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1;r)) \wedge \\ \quad T = \mathit{fix}(\lambda s\cdot\overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \ntriangleright s))) \end{array} \right\}$$

Note that, as discussed in [Jon87], the expression for $T$ is not continuous when $M$ is unboundedly non-deterministic. Since issues concerning unbounded non-determinism are not of interest here, we assume any non-determinism in $M$ is bounded and so we do not require transfinite induction in the proofs below. Thus we take the following rule as the defining property of *fix*

$$\frac{P(\bigcup_{n\geq 0} \mathcal{F}^n(\{\,\}))}{P(\mathit{fix}(\lambda s\cdot\mathcal{F}^n(s))}$$

### 6.7.2 Consistency

For consistency we are required to prove

$$\bigcup_i \mathit{fix}(\lambda r\cdot(\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_i ; r)) = \mathit{fix}(\lambda r\cdot(\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd \bigcup_i M_i ; r))$$

and

$$\bigcap_i \mathit{fix}(\lambda s\cdot\overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} s))) =$$

$$\mathit{fix}(\lambda s\cdot\overline{\neg B} \cup ((\overline{\neg B} \cap \bigcap_i T_1) - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} s)))$$

**Proof**

These follow easily once we establish

$$\bigcup_i \mathit{fix}(\lambda s\cdot\mathcal{G}_i(s)) = \mathit{fix}(\lambda s\cdot \bigcup_i \mathcal{G}_i(s))$$

and

$$\bigcap_i \mathit{fix}(\lambda s\cdot\mathcal{G}_i(s)) = \mathit{fix}(\lambda s\cdot \bigcap_i \mathcal{G}_i(s))$$

which are shown in Appendix A.3.1.

### 6.7.3 Feasibility

Feasibility is exactly the same as in [Jon87].

$$\frac{\begin{array}{c} T_1 \subseteq \mathsf{dom}\, M_1 \\ M = \mathit{fix}(\lambda r\cdot(\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1 ; r)) \\ T = \mathit{fix}(\lambda s\cdot\overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} s))) \end{array}}{T \subseteq \mathsf{dom}\, M}$$

**Proof** *Feasibility*

This follows from lemma W1 of [Jon87].

### 6.7.4 Proof rule

As for conditionals, the treatment of frames adds nothing to the proof rule for iteration other than the requirement that the evaluation of the test be independent of the values of variables outside the read frame.

Our rule for while is equivalent[4] to the one in [Jon87] but with the additional hypothesis, $reads(R, B)$

$$\frac{reads(R,B)}{(F,I,R,W,P,Q^* \wedge P \wedge \neg B) \sqsubseteq \begin{array}{l}\mathsf{while} B\\ \mathsf{do}(F,I,R,W,P \wedge B, Q^* \wedge \overline{P})\\ \mathsf{end}\end{array}}$$

where $Q$ is required to be well-founded and transitive.

### 6.7.5 Soundness

We are required to show

$$\frac{\begin{array}{c} reads(B,R) \\ T = \mathit{fix}(\lambda s \cdot \overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \mathbin{\not\triangleright} s))) \\ M = \mathit{fix}(\lambda r \cdot (\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1 ; r)) \\ drwr_{(F,I,R,W,P \wedge B, Q^* \wedge P)}(T_1, M_1) \end{array}}{drwr_{(F,I,R,W,P,Q^* \wedge P \wedge \neg B)}(T, M)}$$

#### Proof

Each of the four predicates is proved by induction. *dom* and *res* follow immediately from the results in [Jon87], thus it is only in *reads* and *writes* where we show anything new.

**dom** $\overline{P \wedge I} \subseteq T$

Lemma W3 of [Jon87] is

$$\frac{\begin{array}{c} P \wedge B \subseteq T_1 \\ T = \mathit{fix}(\lambda s \cdot \overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \mathbin{\not\triangleright} s))) \\ M = \mathit{fix}(\lambda r \cdot (\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1 ; r)) \end{array}}{P \subseteq T}$$

which gives the required result by substituting $\overline{P \wedge I}$ for $\overline{P}$ since $dom(F, I, R, W, P \wedge B, Q^* \wedge P)$ gives $\overline{P \wedge B \wedge I} \subseteq T_1$.

**res** $\overline{P \wedge I} \lhd M \subseteq \overline{Q^* \wedge P \wedge \neg B}$

Lemma W4 of Jones is

[4] Although Jones' rule looks somewhat different as it is expressed in terms of triples of the form $\{P\}S\{Q\}$, the content of the present rule is the same.

$$\frac{\begin{array}{c}\overline{P \wedge B} \subseteq T_1 \\ \overline{P \wedge B} \lhd M_1 \subseteq \overline{R} \\ \mathsf{rng}\,(\overline{P \wedge B} \lhd M_1) \subseteq \overline{P} \\ T = \mathit{fix}(\lambda s \cdot \overline{\neg B} \cup ((\overline{\neg B} \cap T_1) - \mathsf{dom}\,(M_1 \rhd s))) \\ M = \mathit{fix}(\lambda r \cdot (\overline{\neg B} \lhd \mathit{Id}) \cup (\overline{B} \lhd M_1 ; r))\end{array}}{\overline{P} \lhd M \subseteq R^* \ \wedge \mathsf{rng}\,(\overline{P} \lhd M) \subseteq \overline{P} \ \wedge \mathsf{rng}\,(\overline{P} \lhd M) \subseteq \overline{\neg B}}$$

which again gives the required result by substituting $\overline{P \wedge I}$ for $\overline{P}$ and $\overline{Q \wedge I}$ for $\overline{R}$ since $dom(F, I, R, W, P \wedge B, Q^* \wedge P)$ and $res(F, I, R, W, P \wedge B, Q^* \wedge P)$ give $\overline{P \wedge B \wedge I} \subseteq T$ and $\overline{P \wedge B \wedge I} \lhd M \subseteq \overline{Q \wedge B \wedge I}$.

**writes** $M \subseteq \Xi_{F-W}$

With $M$ and $M_1$ as above, we show

$$\frac{writes(M_1, W)}{writes(M, W)}$$

**Proof** *writes*

We use Scott-induction

$$\frac{\begin{array}{c} pr(\{\,\}) \\ pr(r) \vdash pr(\mathcal{G}(r)) \end{array}}{pr(\mathit{fix}\,\lambda r \cdot \mathcal{G}(r))}$$

with

$$pr(r) \quad \triangleq \quad r \subseteq \Xi_{F-W}$$

and

$$\mathcal{G}(r) \quad \triangleq \quad (\overline{\neg B} \lhd \mathit{Id}) \cup (\overline{B} \lhd M_1 ; r)$$

Now $M = \mathit{fix}(\lambda r \cdot \mathcal{G}(r))$ so the result follows since, trivially we have

$$writes(\{\,\}, W)$$

and furthermore

$$\frac{writes(r, W)}{writes(\mathcal{G}(r), W)}$$

since

$$writes(\mathit{Id}, W) \ \Rightarrow \ writes(\overline{\neg B} \lhd \mathit{Id}, W)$$

and

$$writes(M_1, W) \wedge writes(r, W) \;\Rightarrow\; writes(\overline{B} \lhd M_1;r, W)$$

**readsM** $\Xi_R;M \subseteq M;\Xi_R$

Again with $M$ and $M_1$ as above, we show

$$\frac{\begin{array}{c} readsT(B, R) \\ readsM(M_1, R) \end{array}}{readsM(M, R)}$$

**Proof** *readsM*

The proof follows the same approach. Taking $\mathcal{G}$ as above and

$$pr(r) \;\triangleq\; \Xi_R;r \subseteq r;\Xi_R$$

trivially, we have

$$pr(\{\,\})$$

and furthermore

$$\frac{reads(r, R)}{reads(\mathcal{G}(r), R)}$$

since

$$\begin{array}{rcll}
\Xi_R;\mathcal{G}(r) & = & \Xi_R;((\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1;r)) & \text{defn} \\
 & = & (\Xi_R;\overline{\neg B};Id) \cup (\Xi_R;\overline{B};M_1;r) & \\
 & = & (\overline{\neg B};\Xi_R;Id) \cup (\overline{B};\Xi_R;M_1;r) & \text{by } reads(B, R) \\
 & \subseteq & (\overline{\neg B};Id;\Xi_R) \cup (\overline{B};M_1;\Xi_R;r) & \text{by } reads(M_1, R) \\
 & \subseteq & (\overline{\neg B};Id;\Xi_R) \cup (\overline{B};M_1;r;\Xi_R) & \text{by } reads(r, R) \\
 & = & ((\overline{\neg B} \lhd Id) \cup (\overline{B} \lhd M_1;r));\Xi_R & \\
 & = & \mathcal{G}(r);\Xi_R & \text{defn}
\end{array}$$

**readsT** $\Xi_R (\!|\, T \,|\!) \subseteq T$

With $T$ and $T_1$ as above, we show

$$\frac{\begin{array}{c} reads(B, R) \\ reads(T_1, R) \end{array}}{reads(T, R)}$$

**Proof** *readsT*

By Scott-induction with

$$pr(r) \quad \triangleq \quad \Xi_R \,(\!|\, S \,|\!)\, \subseteq S$$

and

$$\mathcal{G}(r) \quad \triangleq \quad \overline{\neg B} \cup (\overline{\neg B} \cap T_1 - (\mathsf{dom}\, M_1 \not\rhd s))$$

For the base case, note

$$\Xi_R \,(\!|\, \{\,\} \,|\!)\, = \{\,\} \subseteq \{\,\}$$

For the inductive step, we are required to prove

$$\Xi_R \,(\!|\, \mathcal{G}(r) \,|\!)\, \subseteq \mathcal{G}(r)$$

provided $\Xi_R \,(\!|\, r \,|\!)\, \subseteq r$.

Let $\sigma_2 \in \Xi_R \,(\!|\, \mathcal{G}(r) \,|\!)$

so $\exists \sigma_1 \cdot \sigma_1 \Xi_R \sigma_2 \wedge \sigma_1 \in \mathcal{G}(r)$.

Case $\sigma_1 \in \overline{\neg B}$ yields $\sigma_2 \in \overline{\neg B}$ since $\Xi_R \,(\!|\, \overline{\neg B} \,|\!)\, \subseteq \overline{\neg B}$, so $\sigma_2 \in \mathcal{G}(r)$.

Case $\sigma_1 \in \overline{\neg B} \cap T_1 - (\mathsf{dom}\, M_1 \not\rhd s)$ yields $\sigma_1 \in \overline{\neg B} \cap T_1 \wedge M_1 \,(\!|\, \{\sigma_1\} \,|\!)\, \subseteq r$,

so $\sigma_2 \in (\!|\, \overline{\neg B} \,|\!) \wedge \sigma_2 \in (\!|\, T_1 \,|\!)$.

Hence $\sigma_2 \in \overline{\neg B} \wedge \sigma_2 \in T_1$ since $(\!|\, \overline{\neg B} \,|\!)\, \subseteq \overline{\neg B}$ and $(\!|\, T_1 \,|\!)\, \subseteq T_1$, and

$$\begin{array}{rcll}
M_1 \,(\!|\, \{\sigma_2\} \,|\!) & \subseteq & M_1 \,(\!|\, \Xi_R \,(\!|\, \{\sigma_1\} \,|\!) \,|\!) & \\
& = & (\Xi_R ; M_1) \,(\!|\, \{\sigma_1\} \,|\!) & \\
& \subseteq & (M_1 ; \Xi_R) \,(\!|\, \{\sigma_1\} \,|\!) & \text{hypothesis, } \mathit{readsM}(R, M_1) \\
& = & \Xi_R \,(\!|\, M_1 \,(\!|\, \{\sigma_1\} \,|\!) \,|\!) & \\
& \subseteq & \Xi_R \,(\!|\, r \,|\!) & \text{since } M_1 \,(\!|\, \{\sigma_1\} \,|\!)\, \subseteq r \\
& \subseteq & r &
\end{array}$$

so $\sigma_2 \in \overline{\neg B} \cap T_1 - \mathsf{dom}\,(M_1 \not\rhd r)$ as required.

## 6.8 Contract reads and writes

### 6.8.1 Definition

In this case, no new construct is being introduced so there is no semantic definition to make.

### 6.8.2 Proof rules

The rules in Chapter 3 are

$$\frac{W_a \supseteq W_c}{(F, I, R, W_a, P, Q) \sqsubseteq (F, I, R, W_c, P, Q)}$$

$$\frac{R_a \supseteq R_c \supseteq W}{(F, I, R_a, W, P, Q) \sqsubseteq (F, I, R_c, W, P, Q)}$$

### 6.8.3 Soundness: Contract writes

Three out of the four predicates take care of themselves so soundness simply amounts to showing the condition for *writes*

$$\frac{\begin{array}{c} W_a \supseteq W_c \\ M \subseteq \Xi_{F-W_c} \end{array}}{M \subseteq \Xi_{F-W_a}}$$

**Proof** *Contract write frame*

The result follows trivially by noting that:

$$W_a \supseteq W_c \;\Rightarrow\; \Xi_{F-W_a} \subseteq \Xi_{F-W_c}$$

### 6.8.4 Soundness: Contract reads

The rule in Chapter 3 states that it is possible to reduce the read frame provided that the new read frame contains that write frame, i.e. that doing so does not introduce any new write-only variables.

$$\frac{\begin{array}{c} R_a \supseteq R_c \supseteq W \\ writes(F, I, R_c, W, P, Q, (T, M)) \\ reads(F, I, R_c, W, P, Q, (T, M)) \end{array}}{reads(F, I, R_a, W.P, Q, (T, M))}$$

The proviso concerning the write frame is indeed a reasonable requirement as VDM does not recognise the possibility of write-only variables. Note that the proof of soundness of the rule does indeed require this assumption.

**Proof** *Contract read frame*

We will show

$$\frac{\begin{array}{c}\Xi_{R_a} \subseteq \Xi_{R_c} \\ M \subseteq \Xi_{R_a - R_c} \\ \Xi_{R_c} ; M \subseteq M ; \Xi_{R_c}\end{array}}{\Xi_{R_a} ; M \subseteq M ; \Xi_{R_a}}$$

and

$$\frac{\begin{array}{c}\Xi_{R_a} \subseteq \Xi_{R_c} \\ \Xi_{R_c} (\!| T |\!) \subseteq T\end{array}}{\Xi_{R_a} (\!| T |\!) \subseteq T}$$

Then we have the result since

$$R_a \supseteq R_c \;\Rightarrow\; \Xi_{R_a} \subseteq \Xi_{R_c}$$

and

$$M \subseteq \Xi_{F\text{-}W} \subseteq \Xi_{R_a\text{-}R_c}$$

as *writes(W,M)* gives $M \subseteq \Xi_{F-W}$ and so $M \subseteq \Xi_{R_a - R_c}$ since $W \subseteq R_c$.

The second result is trivial since

$$\Xi_{R_a} (\!| T |\!) \subseteq \Xi_{R_c} (\!| T |\!) \subseteq T$$

For the first result, let $(\overleftarrow{\sigma}_1, \sigma_2) \in \Xi_{R_a} ; M$, then we have (writing relations infixed)

$$\exists \overleftarrow{\sigma_2} \cdot \overleftarrow{\sigma_1} \; \Xi_{R_a} \; \overleftarrow{\sigma_2} \; M \; \sigma_2$$

hence $\overleftarrow{\sigma_1} \; \Xi_{R_c} \; \overleftarrow{\sigma_2} \; M \; \sigma_2$ since $\Xi_{R_a} \subseteq \Xi_{R_c}$ (hypothesis 1).

So by hypothesis 3, we have $(\overleftarrow{\sigma}_1, \sigma_2) \in M ; \Xi_{R_c}$ so

$$\exists \sigma_1 \cdot \overleftarrow{\sigma_1} \; M \; \sigma_1 \; \Xi_{R_c} \; \sigma_2$$

Now it remains to show that $\sigma_1 \; \Xi_{R_a - R_c} \; \sigma_2$ which follows since

$$\sigma_1 \;\; M^{\text{-}1} \;\; \overleftarrow{\sigma_1} \;\; \Xi_{R_a} \;\; \overleftarrow{\sigma_2} \;\; M \;\; \sigma_2$$

and $M \subseteq \Xi_{R_a\text{-}R_c}$, (hypothesis 2), and $\Xi_{R_a} \subseteq \Xi_{R_a\text{-}R_c}$.

As stated earlier, this is the one place where the four conditions cannot be proved independently and it is interesting to investigate why this should be. We will show in the next chapter (Section 7.2) that, with either of the stricter "must-write" or "may-write" interpretations described in Section 5.2.1, the rule can be proved without requiring the write frame constraint.

### 6.8.5 Corollary

These two results immediately give rise to the following corollaries.

Let

$$R_{\cup} \quad \triangleq \quad R_1 \cup R_2 \quad \text{and} \quad R_{\cap} \quad \triangleq \quad R_1 \cap R_2$$

and define $W_{\cap}$ and $W_{\cup}$ similarly. Then we have

$$mk\text{-}OpDef(F, I, R_{\cup}, W, P, Q) \sqsubseteq mk\text{-}OpDef(F, I, R_i, W, P, Q), \;\; i = 1, 2$$

$$mk\text{-}OpDef(F, I, R, W_{\cup}, P, Q) \sqsubseteq mk\text{-}OpDef(F, I, R, W_i, P, Q), \;\; i = 1, 2$$

$$mk\text{-}OpDef(F, I, R_i, W, P, Q) \sqsubseteq mk\text{-}OpDef(F, I, R_{\cap}, W, P, Q), \;\; i = 1, 2$$

$$mk\text{-}OpDef(F, I, R, W_i, P, Q) \sqsubseteq mk\text{-}OpDef(F, I, R, W_{\cap}, P, Q), \;\; i = 1, 2$$

provided $R_i \supseteq W_i$ as appropriate.

Furthermore we also have that

$$[\![mk\text{-}OpDef(F, I, R_{\cap}, W, P, Q)]\!] = \bigcap_{i=1,2} [\![mk\text{-}OpDef(F, I, R_i, W, P, Q)]\!]$$

$$[\![mk\text{-}OpDef(F, I, R, W_{\cap}, P, Q)]\!] = \bigcap_{i=1,2} [\![mk\text{-}OpDef(F, I, R, W_i, P, Q)]\!]$$

The proof of which can be found in Appendix A.3.2. However note that, in general, unioning of frames does not distribute through operation definitions.

## 6.9 Manipulating the overall frame

The following two rules for manipulating the overall frame are also given in Chapter 3

**Contract Frame**

$$\frac{F_A \overset{ind}{\supseteq} F_C, \; (F_C, I_{F_C}, R, W, P, Q) \colon OpDef}{(F_A, I, R, W, P, Q) \sqsubseteq (F_C, I_{F_C}, R, W, P, Q)}$$

**Expand Frame**

$$\frac{F \cap S = \phi}{(F, I, R, W, P, Q) \sqsubseteq \mathsf{var}\ \mathsf{S}\ \mathsf{in}\ (F \cup S, I, R \cup S, W \cup S, P, Q)\ \mathsf{end}}$$

However, the treatment given thus far has assumed a fixed overall frame and so these rules cannot yet be justified. In fact the manipulation of the overall frame involves quite different issues from those discussed here and so its treatment is deferred to the next chapter (Section 7.4).

## 6.10 Non-interference

We now consider again the non-interference result from Chapter 4, in particular we are now in a position to prove the stronger form of the theorem which states that non-interference is preserved by refinement. Recall that for two operations on the same context, $F$ and $I$,

$$Op_i \quad \triangleq \quad mk\text{-}OpDef(F, I, R_i, W_i, P_i, Q_i), \quad i = 1, 2$$

if

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

and if each $Op_i$ is refined by $Op'_i$

$$Op_i \sqsubseteq Op'_i$$

then non-interference states that we have

$$Op'_1 ; Op'_2 = Op'_2 ; Op'_1$$

A proof of this is given in detail in Appendix A.3.4. It relies on the four 'frame' properties of the semantics

$$reads(R_1, M_1), \quad writes(W_1, M_1), \quad reads(R_2, M_2), \quad writes(W_2, M_2)$$

the non-interference conditions in a slightly different form

$$F - W_2 \supseteq R_1 \text{ and } F - W_1 \supseteq R_2$$

and on the 'global' conditions

$$R_1 \supseteq W_1 \text{ and } R_2 \supseteq W_2$$

However, it could reasonably be argued that non-interference should hold provided the write frame of each operations does not intersect either frame of the other operation, irrespective of whether it is contained in its own read frame. The alternative definition of *reads* developed in Section 7.2 also yields this stronger form of the non-interference theorem.

## 6.11 Weaking the invariant, reads and writes

To conclude this chapter, recall that two possibilities were discussed in Chapter 3 (Section 3.3) concerning the interpretation of the invariant, reads and writes. A "transient" interpretation simply required that the relevant conditions hold for the before and after states, whereas a "persistent" interpretation required that the relevant condition be maintained throughout execution.

Although the latter interpretation was advocated in Chapter 3, in fact, it is the former alternative that is being modelled by the present semantics. This arises because this model of sequential composition imposes no constraint on the intermediate state since the existential quantification in the definition of sequencing removes all record of that state.

In terms of the non-interference theorem given above, although this interpretation suffices to ensure non-interference where the two operations are sequenced in either order, it is not strong enough to allow a finer granularity of interleaving where fragments of each operation are executed alternately. That is, it requires atomicity at the level of the entire operation.

If we are to model a persistent interpretation of the invariant and frames, we must further enrich the semantic model of operations. In the next chapter (Section 7.3) we construct a semantics where all sequences of states visited during execution are modelled, thus completing a trichotomy of increasingly concrete semantic models.

# Chapter 7

# Further Extensions to the Semantics

*"The 'real' mathematics of the 'real' mathematicians, the mathematics of Fermat and Euler and Gauss and Abel and Riemann, is almost wholly 'useless'."*

## 7.1 Introduction

This chapter considers three issues not addressed by the semantics of Chapter 6. The first, which arose in Section 6.8.4, is the non-independence of the treatment of the read and write frames in the proof of soundness of the contract read rule. This concerns the introduction of write-only variables and is discussed in Section 7.2. The second concerns the modelling of the intermediate states visited during the execution of an operation described in Section 6.11. This is discussed in Section 7.3. The third, concerning the manipulation of the overall frame (Section 6.9) is discussed in Section 7.4.

## 7.2 Introducing write-only variables

It was observed in Section 6.8.4 that the proof of soundness for contracting the read frame required that the reads should contain the writes for each operation. Even though this is the case for VDM operations, it may be surprising that this assumption is required in the proof since it is reasonable to expect that any implementation reading fewer variables than permitted should be valid whatever the write frame might be. Furthermore, this is the only place where the orthogonality of proof of the four predicates is broken.

Relaxing the constraint leads to the possibility of introducing write-only variables during refinement which gives another motivation for their interpretation.

This section will show that for both the must-write and may-write interpretations of write-only variables given in Chapter 5, the contract reads rule could have been proven without recourse to the troublesome hypothesis. It then goes on to show that with a slight adjustment to its definition, the secure semantics can also yield independence from the write frame when contracting the reads.

### 7.2.1 Introducing must-write variables

With the must-write interpretation of write-only variables, the following property holds of the read constraint without the requirement that $R_i \supseteq W$

$$\frac{\begin{array}{c} R_a \supseteq R_c \\ reads_{must}(R_c, M) \end{array}}{reads_{must}(R_a, M)}$$

where

$$\begin{array}{lll} reads_{must}(R, M) & \triangleq & \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_R \overleftarrow{\sigma_2} \;\Rightarrow \\ & & \qquad\qquad\qquad\qquad \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2 \wedge \sigma_1 \Xi_W \sigma_2 \end{array}$$

This immediately yields a correspondingly stronger rule for contracting the read frame

$$\frac{R_a \supseteq R_c}{(F, I, R_a, W, P, Q) \sqsubseteq (F, I, R_c, W, P, Q)}$$

The proof is trivial since $R_a \supseteq R_c$ gives $\overleftarrow{\sigma_1} \Xi_{R_a} \overleftarrow{\sigma_2} \;\Rightarrow\; \overleftarrow{\sigma_1} \Xi_{R_c} \overleftarrow{\sigma_2}$, and $\Xi_R$ only appears on the left hand side of the implication in the definition.

Note, however, that we have not really achieved the independence from $W$ which was sought since this condition still depends on $W$ through the appearance of $\overleftarrow{\sigma_i} \xrightarrow{\mathrm{W}} \sigma_i$ and $\sigma_1 \Xi_W \sigma_2$.

### 7.2.2 Introducing may-write variables

The stronger rule for contract reads can also be proved with the may-write semantics, but first we need to give this interpretation a formal definition which was omitted in Chapter 5.

For the may-write interpretation, we require that each variable may either be left unchanged or may be overwritten with a value depending only on the reads. To define this, we have to consider each variable separately. This is done by expanding $\sigma_1 \Xi_W \sigma_2$ as[1] $\forall x \in W \cdot \sigma_1 \Xi_{\{x\}} \sigma_2$

Thus we get

$$\begin{array}{l} reads_{may}(R, M) \;\triangleq \\ \qquad \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \; \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_R \overleftarrow{\sigma_2} \;\Rightarrow \\ \qquad\qquad\qquad\qquad\quad \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2 \wedge \forall x \in W \cdot \sigma_2 \Xi_{\{x\}} \sigma_1 \vee (\overleftarrow{\sigma_1} \Xi_{\{x\}} \sigma_1 \wedge \overleftarrow{\sigma_2} \Xi_{\{x\}} \sigma_2) \end{array}$$

[1]Note that, strictly speaking, the "$\forall x \in W \cdot \ldots$" is a meta-notational shorthand for a conjunction of clauses, one for each variable in $W$.

Now the proof of soundness for this semantics is identical to that for *must* since again the right hand side of the implication does not contain a mention of $R$.

### 7.2.3 Comparing the three interpretations

Note that it is only for the write-only variables that the three forms lead to a different interpretation. To see this write

$$\begin{array}{lll} A & \triangleq & \sigma_2 \Xi_{\{x\}} \sigma_1 \\ B & \triangleq & \overleftarrow{\sigma_1} \Xi_{\{x\}} \sigma_1 \wedge \overleftarrow{\sigma_2} \Xi_{\{x\}} \sigma_2 \end{array}$$

and note that all three interpretations are of the form

$$\forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \ \overleftarrow{\sigma_1} \xrightarrow{\mathrm{W}} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_R \overleftarrow{\sigma_2} \ \Rightarrow \ \overleftarrow{\sigma_2} \xrightarrow{\mathrm{W}} \sigma_2 \wedge P$$

where

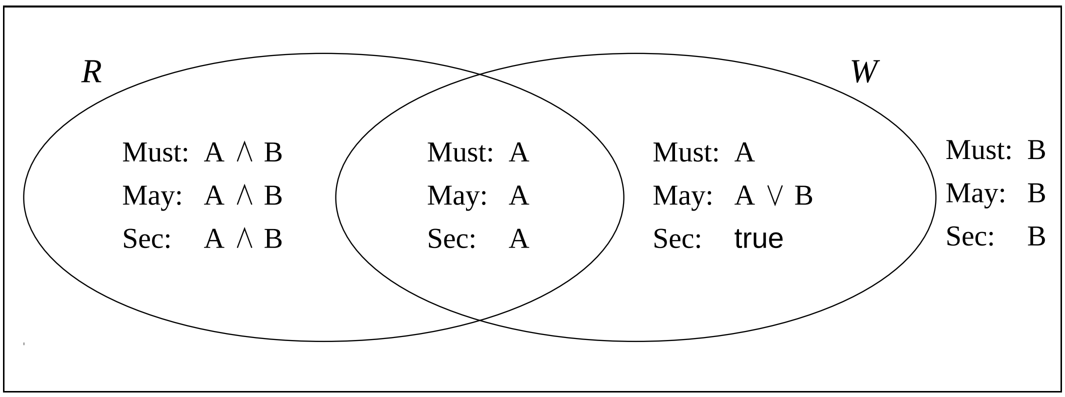
$$\begin{array}{ll} P = \forall x \in W \cdot A & \text{for } \mathit{must}, \\ P = \forall x \in W \cdot A \vee B & \text{for } \mathit{may}, \\ P = \forall x \in R \cdot A & \text{for } \mathit{secure}. \end{array}$$

Now, in each case, we can rewrite $P$ as follows

$$\begin{array}{lllll} \text{Must:} & \forall x \notin W \cdot B & \wedge \ \forall x \in R \cdot A & \wedge \ \forall x \in W - R \cdot A \\ \text{May:} & \forall x \notin W \cdot B & \wedge \ \forall x \in R \cdot A & \wedge \ \forall x \in W - R \cdot A \vee B \\ \text{Secure:} & \forall x \notin W \cdot B & \wedge \ \forall x \in R \cdot A & \wedge \ \forall x \in W - R \cdot \mathsf{true} \end{array}$$

since, in all cases, for $x \notin W$ we have $B$, and for $x \in R$ we have $\overleftarrow{\sigma_1} \Xi_{\{x\}} \overleftarrow{\sigma_2}$ so $B \Rightarrow A$. Thus, for $x \in R - W$, we get $A \wedge B$, and, for $x \in RW$ we get $A$ for all three interpretations[2]. Whereas, for $x \in WO$, we get: $A$ for *must*, $A \vee B$ for *may* and $\mathsf{true}$ for *secure*.

This can be pictured as




[2]For *may*, note that $B \wedge (B \Rightarrow A) \wedge (A \vee B) \Leftrightarrow A \wedge B$, and also that $(B \Rightarrow A) \wedge (A \vee B) \Leftrightarrow A$.

So we see that *must* is stronger than *may* which in turn is stronger than *secure*. This leads us to try to strengthen the secure interpretation to regain the independence from the write frame in the proof.

### 7.2.4 Introducing secure variables

We now develop a variant of the 'secure' interpretation of write-only variables for which any contraction of the read frame is a refinement, whatever the write frame may be.

First note that this is not the case for the previous version since an implementation with a small read frame may exchange information between two variables which are both unwritten, whereas only one of those variables may be included in the larger read frame of the specification. This is demonstrated by the following example for which the abstract operation, $Op_a$, has no read respecting implementation, whereas the concrete operation, $Op_c$, which is identical except for a smaller read frame, does have.

$$
\begin{array}{ll}
\begin{array}{l}
Op_a \\
\textsf{ext rd } x \quad\; : \mathbb{N} \\
\quad\;\; \textsf{wr } x, y : \mathbb{N} \\
\textsf{post } x = \overleftarrow{y} \wedge y = 0
\end{array}
&
\begin{array}{l}
Op_c \\
\textsf{ext rd } \{\,\} \\
\quad\;\; \textsf{wr } x, y : \mathbb{N} \\
\textsf{post } x = \overleftarrow{y} \wedge y = 0
\end{array}
\end{array}
$$

Note that $Op_a$ is unimplementable since information from the unread $\overleftarrow{y}$ is "leaking" into $x$ which is read, however this is permitted for $Op_c$ since neither variable is read. (Details are given in Appendix A.4.1.)

In attempting to derive an alternative interpretation which does allow the introduction of write-only variables and hence the stronger form of contracts reads rule, we must ensure that the original value of each unread variable can only affect its own final value and not the final values of any other unread variable. Then, if some other unread variable falls into the read scope, the read constraint will not be broken.

A first attempt to define such a strengthening of the condition might be to move the conjunction over $R$ outside the implication to give[3]

$$
\begin{array}{l}
reads_{secure_1}(R, M) \quad \triangleq \\
\qquad \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \forall x \in R \cdot \;\; \overleftarrow{\sigma_1} \overset{\circ}{\longrightarrow} \sigma_1 \wedge \overleftarrow{\sigma_1}\, \Xi_{\{x\}} \overleftarrow{\sigma_2} \;\; \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad\;\; \overleftarrow{\sigma_2} \overset{\circ}{\longrightarrow} \sigma_2 \wedge \sigma_1 \Xi_{\{x\}} \sigma_2
\end{array}
$$

However this does not give the required property since this condition gets weaker as $R$ gets smaller, which is precisely the opposite of what is sought.

A second attempt might be to require that for each unread variable, any two states that differ only in the value of that unread variable, can only lead to result states that differ in the value of the same variable.

[3]using $(\forall x \cdot P(x) \;\Rightarrow\; Q(x)) \;\Rightarrow\; (\forall x \cdot P(x) \;\Rightarrow\; \forall x \cdot Q(x))$

$$
\begin{array}{l}
reads_{secure_2}(R, M) \quad \triangleq \\
\quad \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \forall x \in F - R \cdot \; \overleftarrow{\sigma_1} \overset{\circ}{\longrightarrow} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_{F-\{x\}} \overleftarrow{\sigma_2} \;\Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \overleftarrow{\sigma_2} \overset{\circ}{\longrightarrow} \sigma_2 \wedge \sigma_1 \Xi_{F-\{x\}} \sigma_2
\end{array}
$$

Now this condition clearly strengthens as $R$ contracts, and indeed does yield the write independence property we seek, but it is actually weaker than the original and, in fact is too weak as it does not prohibit some operations that do require reading outside the read frame. (An example is given in Appendix A.4.2)

Rather the correct formulation is the following:

$$
\begin{array}{l}
reads_{secure_3}(R, M) \quad \triangleq \\
\quad \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \forall S \supseteq R \cdot \; \overleftarrow{\sigma_1} \overset{\circ}{\longrightarrow} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_S \overleftarrow{\sigma_2} \;\Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \overleftarrow{\sigma_2} \overset{\circ}{\longrightarrow} \sigma_2 \wedge \sigma_1 \Xi_S \sigma_2
\end{array}
$$

which is clearly stronger than *secure* in general, but yields an equivalent interpretation when $R \supseteq W$ (since for $x \in S - R$ we have $x \notin W$ and so $\sigma_2 \; \Xi_{\{x\}} \; \overleftarrow{\sigma_2} \; \Xi_{\{x\}} \; \overleftarrow{\sigma_1} \; \Xi_{\{x\}} \; \sigma_1$). Furthermore, it clearly gets stronger as $R$ gets smaller and so is preserved by contracting the read frame, whatever the write frame may be.

This condition ensures that there is no 'leaking' of information between the unread variables, since for example, if $x$ and $y$ are both unread, then any $S$ which contains $x$ but not $y$ will ensure that there is no leak from $x$ to $y$. Thus it yields the stronger form of the contract reads rule which does not require $R \supseteq W$.

This interpretation also maintains the independence from the write frame in its definition and so it could be used as a basis for an orthogonal definition of the semantics of operations as developed in Chapter 5. However, it is considerably more unwieldy that the definition used there.

### 7.2.5 Alternative forms for this condition

Having given an alternative interpretation for write-only variables, we explore this definition a little further by giving two criteria equivalent to it and a cautionary example demonstrating that a third form is not equivalent.

The first equivalent form shows that we can consider each of the unread variables separately

$$
\begin{array}{l}
reads_{secure'_3}(R, M) \quad \triangleq \\
\quad \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \; \Xi_R ; M \subseteq M ; \Xi_R \wedge \\
\qquad\qquad\qquad\qquad\qquad \forall x \in F - R \cdot \; \overleftarrow{\sigma_1} \overset{\circ}{\longrightarrow} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_{R \cup \{x\}} \overleftarrow{\sigma_2} \;\Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \overleftarrow{\sigma_2} \overset{\circ}{\longrightarrow} \sigma_2 \wedge \sigma_1 \Xi_{R \cup \{x\}} \sigma_2
\end{array}
$$

The second form shows that the $\sigma_2$ need not be chosen independently of the particular $S$

$$
\begin{array}{l}
reads_{secure_3''}(R, M) \quad \triangleq \\
\qquad \forall S \supseteq R \cdot \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \quad \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \land \overleftarrow{\sigma_1} \Xi_S \overleftarrow{\sigma_2} \;\; \Rightarrow \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \land \sigma_1 \Xi_S \sigma_2
\end{array}
$$

The equivalences are proved in Appendix A.4.3 by showing the chain of implications

$$reads_{secure_3} \;\Rightarrow\; reads_{secure_3'} \;\Rightarrow\; reads_{secure_3''} \;\Rightarrow\; reads_{secure_3}$$

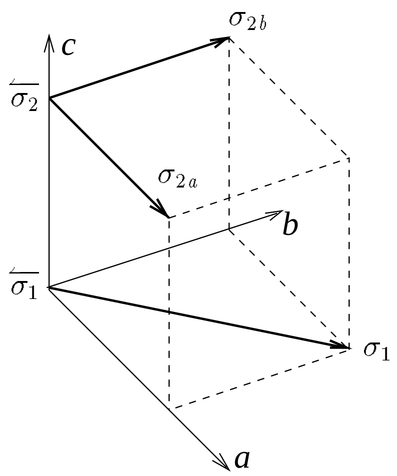
Note, however, that the universal quantification over $F - R$ cannot be moved outwards in $reads_{secure_3'}$, to give

(1) $\quad \forall x \in F - R \cdot \forall \overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1 \cdot \exists \sigma_2 \cdot \ldots$

since in that case we get a strictly weaker predicate. This is shown by the following example

$$
\begin{array}{llll}
\Sigma & :: & a & : \{0,1\} \\
 & & b & : \{0,1\} \\
 & & c & : \{0,1\}
\end{array}
$$

*Op*
ext rd $\{\,\}$
    wr $a, b : \{0,1\}$
pre $a = b = 0$
post $a + b + c = 2$




Here, we have (1) since given

$$\overleftarrow{\sigma_1} = (0,0,0), \overleftarrow{\sigma_2} = (0,0,1), \sigma_1 = (1,1,0)$$

(all other cases are trivial), letting

$$\sigma_{2a} = (1,0,1), \sigma_{2b} = (0,1,1)$$

then, if $x$ is given before $\sigma_2$ is chosen, say $x = a$, we can choose $\sigma_2 = \sigma_{2a}$, giving $\sigma_2 \Xi_{\{x\}} \sigma_1$ as required. But $reads_{secure_3'}$ does not hold since a suitable $\sigma_2$ cannot be chosen before $x$ is known.

Finally, note that the second alternative form yields a concise expression in the spirit of Section 5.3 as follows

$$reads_{secure_3} \;\Leftrightarrow\; \forall S \subseteq R \cdot \Xi_S ; M \subseteq M ; \Xi_S$$

This recovers the elegance[4] of the condition used in Chapters 5 and 6 and, since $R$ can be treated as a parameter to the definitions and proofs given there, also yields the earlier results.

### 7.2.6 Non-interference

It is shown in Appendix A.4.4 that this interpretation yields the non-interference theorem without recourse to the assumptions $R_i \supseteq W_i$. Thus, as discussed in Section 6.10, we have

$$\frac{\begin{array}{c} reads(R_1, M_1) \wedge writes(W_1, M_1) \\ reads(R_2, M_2) \wedge writes(W_2, M_2) \\ W_1 \cap (R_2 \cup W_2) = \{\} = W_2 \cap (R_1 \cup W_1) \end{array}}{M_1 ; M_2 = M_2 ; M_1}$$

## 7.3 Modelling intermediate states

Recall from the proof theory in Chapter 3 that, since we are modelling the behaviour of code which syntactically satisfies the read and write access conditions, we wish to give a persistent interpretation to the frames. That is, not only must the before and after states of the operation satisfy the read and write constraints, but so must any intermediate states visited during the execution.

As it stands, the semantic model in Chapter 6 does not support this interpretation as there are no constraints on the middle state in sequential composition which is 'quantified away' via the definition of relational composition. So, although the rules in Chapter 3 are sound with respect to this model, they are incomplete in that they cannot be used to justify all refinements. However, it is the proof theory, rather than the semantics, which yields the desired set of refinements. So, instead of resolving this incompleteness by adding rules to the proof theory to allow more refinements, we enrich the semantics to permit fewer.

This section investigates how this can be done by modelling the intermediate states of an operation execution, thus interpreting operations as sets of sequences of states rather than sets of pairs. As in Chapters 5 and 6, we give two models for operations. The first model is a set of possible executions (this time each execution is a sequence of states), the second is a set of such sets. These models 'reify' the earlier two and justify the omission of rules for weaking the read and write constraints from the proof theory.

Note that, in taking this approach, we are seeking to model the read and write accesses of code without the overhead of the heavier semantic models required to model shared-state parallelism in general. The cost of this approach is that we cannot model the interleaving of operation steps since the behaviour of an operation under interference is non-extensional in that it cannot be deduced from the uninterrupted behaviour. A fuller

[4] This formulation also bears an even closer syntactic resemblance to the definition of bisimulation although this is probably merely coincidence.

treatment would involve the definition of interleaving based on a richer model for shared-state parallelism such as that of resumptions in [Jon91], however, the simpler approach suffices for present purposes.

**Modelling non-termination.** Note that, as in the relational semantics, there is a choice in how one models non-termination. One possibility is to give an explicit termination set in the interpretation as was done for relational semantics. However, in the present setting, such an approach is undesirable as it loses information concerning the states visited during the non-terminating execution. So here we choose the more direct interpretation which models non-terminating executions as infinite sequences. This corresponds to the alternative relational model mentioned in Section 5.5.1 which 'lifts' the result state space to include a value representing non-termination. The use of infinite sequences to model non-termination has the advantage that it simplifies that treatment of termination under sequencing however, it does introduce an extra degree of cardinality.

**Modelling inaction.** There is also a choice in the modelling of inaction which is reflected in the interpretation of skip. One possibility is to model it as a 'stuttering step', that is, two identical consecutive elements in the sequence. Another is simply to regard it as a non-event leading to sequences with no consecutive repeats. Similarly, terminating executions can be modelled by finite sequences or by infinite ones which repeat from some point onwards (cf. [Lam91]). Here we will find it convenient to insist that sequences are non-stuttering and so use finite sequences for terminating executions.

### 7.3.1 Notation for finite and infinite sequences

We first introduce some notation for handling finite and infinite non-empty, non-repeating sequences.

Let

$\Sigma^f$ be the finite, non-empty, non-stuttering sequences of elements from $\Sigma$,

$\Sigma^i$ be the infinite, non-empty, non-stuttering sequences of elements from $\Sigma$,

$\Sigma^a = \Sigma^i \cup \Sigma^f$ be all the above.

Let $N : \mathcal{P}\Sigma^a$, be a set of sequences and write[5]

$N^f$ to mean the set of finite elements of $N$

$N^i$ to mean the set of infinite elements of $N$

[5] There is a slight ambiguity in the notation here as a superscript on $\Sigma$ has a different meaning from one on $N$, however, the meaning will always be clear in context.

We will also use decorated variable names as follows

$\sigma^f$ for a finite element of $\Sigma^a$

$\sigma^i$ for an infinite element of $\Sigma^a$, and

$\sigma^a$ for an arbitrary element of $\Sigma^a$.

Now $\sigma^a : \Sigma^a$ will be used to model a possible execution of a machine, and $N : \mathcal{P}\Sigma^a$ to model a set of possible executions.

Indexing will start with 1 and, for finite sequences, will end with $\mathsf{len}\,\sigma^f$.

As sequences are non-empty, head is total over all sequences, and last over finite sequences

$$\mathsf{hd} : \Sigma^a \to \Sigma$$
$$\mathsf{hd}(\sigma^a) \triangleq \sigma^a(1)$$

$$\mathsf{lt} : \Sigma^f \to \Sigma$$
$$\mathsf{lt}(\sigma^f) \triangleq \sigma^f(\mathsf{len}\,\sigma^f)$$

Define domain and domain restriction over sets of sequences by[6]

$$\mathsf{dom} : \mathcal{P}\Sigma^a \to \mathcal{P}\Sigma$$
$$\mathsf{dom}(N) \triangleq \{\mathsf{hd}\,\sigma^a \mid \sigma^a \in N\}$$

$$\_ \lhd \_ : \mathcal{P}\Sigma \times \mathcal{P}\Sigma^a \to \mathcal{P}\Sigma^a$$
$$S \lhd N \triangleq \{\sigma^a \in N \mid \mathsf{hd}\,\sigma^a \in S\}$$

For $N : \mathcal{P}\Sigma^a$, define some functions that will be useful to recover the relational model

$$\mathcal{T}(N) \triangleq \Sigma - \mathsf{dom}\,N^i$$

$$\mathcal{M}(N) \triangleq \{(\mathsf{hd}\,\sigma^f, \mathsf{lt}\,\sigma^f) \mid \sigma^f \in N^f\}$$

$$\mathcal{A}_{i\text{-}1}(N) \triangleq \{(\sigma^a(i\text{-}1), \sigma^a(i)) \mid \sigma^a \in N \wedge i \in \mathsf{inds}\,\sigma^a - \{1\}\}$$

$$\mathcal{A}(N) \triangleq \bigcup_{i \in \mathbb{N}_1 - \{1\}} \mathcal{A}_{i\text{-}1}(N)$$

Note that any singleton sequences in $N$ become pairs in $\mathcal{M}$, but make no contribution to $\mathcal{A}_i$ or $\mathcal{A}$.

The following properties can easily be deduced from the above. For $\mathcal{S}_N : \mathcal{P}\mathcal{P}\Sigma^a$ and $N_i \in \mathcal{S}_N$ we have

$$\bigcup_{N_i \in \mathcal{S}_N} (N_i^i) = (\bigcup_{N_i \in \mathcal{S}_N} N_i)^i \qquad \bigcup_{N_i \in \mathcal{S}_N} (N_i^f) = (\bigcup_{N_i \in \mathcal{S}_N} N_i)^f$$

[6] This should not be confused with domain used on individual sequences where sequences are considered to be maps from natural numbers to elements and domain delivers the indices of a sequence.

$$\bigcap_{N_i \in \mathcal{S}_N} \mathcal{T}(N_i) = \mathcal{T}(\bigcup_{N_i \in \mathcal{S}_N} N_i) \qquad \bigcup_{N_i \in \mathcal{S}_N} \mathcal{M}(N_i) = \mathcal{M}(\bigcup_{N_i \in \mathcal{S}_N} N_i)$$

$$\bigcup_{N_i \in \mathcal{S}_N} \mathcal{A}_i(N_i) = \mathcal{A}_i(\bigcup_{N_i \in \mathcal{S}_N} N_i) \qquad \bigcup_{N_i \in \mathcal{S}_N} \mathcal{A}(N_i) = \mathcal{A}(\bigcup_{N_i \in \mathcal{S}_N} N_i)$$

$$P \lhd \mathcal{M}(N) = \mathcal{M}(P \lhd N) \qquad \bigcup_{N_i \in \mathcal{S}_N} (P \lhd N_i) = P \lhd \bigcup_{N_i \in \mathcal{S}_N} (N_i)$$

Appendix A.4.7 gives further definitions of operators on sequences as well as some algebraic properties of them.

### 7.3.2 Interpreting pre-post specifications

A specification is to be modelled as the set of all possible execution sequences. So for the implicit operation, $mk\text{-}OpDef(F, I, R, W, P, Q)$, and for any set of sequences, $N\colon \mathcal{P}\Sigma^a$, we define some compliance conditions similar to those of Chapter 5.

$$dom_{c(F,I,R,W,P,Q)}(N) \;\triangleq\; \overline{P \wedge I} \subseteq \mathcal{T}(N)$$

$$res_{c(F,I,R,W,P,Q)}(N) \;\triangleq\; \mathcal{M}(\overline{P \wedge I} \lhd N) \subseteq \overline{Q \wedge I}$$

$$writes_{c(F,I,R,W,P,Q)}(N) \;\triangleq\; \mathcal{A}(N) \subseteq \Xi_{F-W}$$

Note that $res_c$ gives a transient interpretation to the invariant as it does not constrain the intermediate states. A persistent interpretation can be given by adding the condition

$$inv_{c(F,I,R,W,P,Q)}(N) \;\triangleq\; \mathcal{A}(\overline{P \wedge I} \lhd N) \subseteq \overline{I} \times \overline{I}$$

On the other hand, *writes* is expressed as a constraint on each step and thus is persistent. Transitivity ensures the property holds for the operation as a whole.

As ever, the remaining task is to define $reads_{c(F,I,R,W,P,Q)}(N)$ which will be done in Section 7.3.4, but first we see how these conditions are used to give the semantic models. Following Chapters 5 and 6, let

$$\mathcal{S}_N = \left\{ N\colon \mathcal{P}\Sigma^a \;\middle|\; \begin{array}{ll} & dom_{c(F,I,R,W,P,Q)}(N) \\ \wedge & res_{c(F,I,R,W,P,Q)}(N) \\ \wedge & writes_{c(F,I,R,W,P,Q)}(N) \\ \wedge & reads_{c(F,I,R,W,P,Q)}(N) \end{array} \right\}$$

and define two interpretations, the first being this set of sets of sequences

$$[\![Op]\!]_N \quad \triangleq \quad \mathcal{S}_N$$

and the second being the most general of these

$$[\![Op]\!]_{N_0} \quad \triangleq \quad \bigcup_{N_i \in \mathcal{S}_N} N_i$$

Define a retrieve function

$$retr_c : \mathcal{P}\mathcal{P}\Sigma^a \to \mathcal{P}\Sigma^a$$
$$retr_c(\mathcal{S}) \quad \triangleq \quad \bigcup_{N_i \in \mathcal{S}_N} N_i$$

so that we have

$$retr_c([\![Op]\!]_N) = [\![Op]\!]_{N_0}$$

Clearly, there are some well-definedness conditions to show of these models, as well as consistency with the previous models. These are considered in the next section.

### 7.3.3 Reduction to relational semantics

The two models in the last section easily reduce to those in Chapters 5 and 6 by the following retrieve functions

$$retr_o : \mathcal{P}\Sigma^a \to \mathcal{P}\Sigma \times \mathcal{P}(\Sigma \times \Sigma)$$
$$retr_o(N) \quad \triangleq \quad (\mathcal{T}(N), \mathcal{M}(N))$$

and

$$retr_{\mathcal{S}} : \mathcal{P}\mathcal{P}\Sigma^a \to \mathcal{P}(\mathcal{P}\Sigma \times \mathcal{P}(\Sigma \times \Sigma))$$
$$retr_{\mathcal{S}}(\mathcal{S}_N) \quad \triangleq \quad \{(\mathcal{T}(N), \mathcal{M}(N)) \mid N \in \mathcal{S}_N\}$$

Thus we have four semantic models and four retrieve functions between them as indicated in Figure 7.1. The expected commutativity property follows from the consistency of each retrieval which is defined as follows.

#### Well-definedness

To demonstrate that these four models and retrievals are well-defined we need to show that for each retrieve function $f : \mathcal{D}_c \to \mathcal{D}_a$ we have

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
(T,M){:}\,\mathcal{P}\Sigma \times \mathcal{P}\Sigma^2 & & \mathcal{S}{:}\,\mathcal{P}(\mathcal{P}\Sigma \times \mathcal{P}\Sigma^2) \\
\llbracket Op \rrbracket_o & & \llbracket Op \rrbracket \arrow[ll, "(\bigcap_{(T,M)\in\mathcal{S}} T\,,\ \bigcup_{(T,M)\in\mathcal{S}} M)"', "retr"] \\
\llbracket Op \rrbracket_{N_0} \arrow[u, "(\mathcal{T}(N),\mathcal{M}(N))", "retr_o"'] & & \llbracket Op \rrbracket_N \arrow[ll, "retr_c"', "\bigcup_{N_i\in\mathcal{S}_N} N_i"] \arrow[u, "retr_{\mathcal{S}}", "\{(\mathcal{T}(N_i),\mathcal{M}(N_i)) \mid N_i\in\mathcal{S}_N\}"'] \\
N{:}\,\mathcal{P}\Sigma^a & & \mathcal{S}_N{:}\,\mathcal{P}\mathcal{P}\Sigma^a
\end{tikzcd}
$$

Figure 7.1: Four semantic models for $Op$ and retrievals between them.

1) $f$ is a total surjection,

2) $f$ is consistent. i.e. for all $Op$, $f\llbracket Op \rrbracket_{\mathcal{D}_c} = \llbracket Op \rrbracket_{\mathcal{D}_a}$,

3) $f$ preserves refinement, i.e. $Op_1 \sqsubseteq_{\mathcal{D}_c} Op_1 \ \vdash \ Op_1 \sqsubseteq_{\mathcal{D}_a} Op_1$.

More detail is given in Appendix A.4.5.

#### Feasibility

Feasibility in this model is given by the condition

$$\mathsf{dom}\, N = \Sigma$$

and it is shown in Appendix A.4.6 that this is consistent with the $(T, M)$ version, i.e.

$$\mathsf{dom}\, N = \Sigma \ \Leftrightarrow \ \mathcal{T}(N) \subseteq \mathsf{dom}\, \mathcal{M}(N)$$

### 7.3.4 Definition of non-reading for sequences

We now define $reads_{c(F,I,R,W,P,Q)}(N)$ for this semantic model which was postponed from Section 7.3.2.

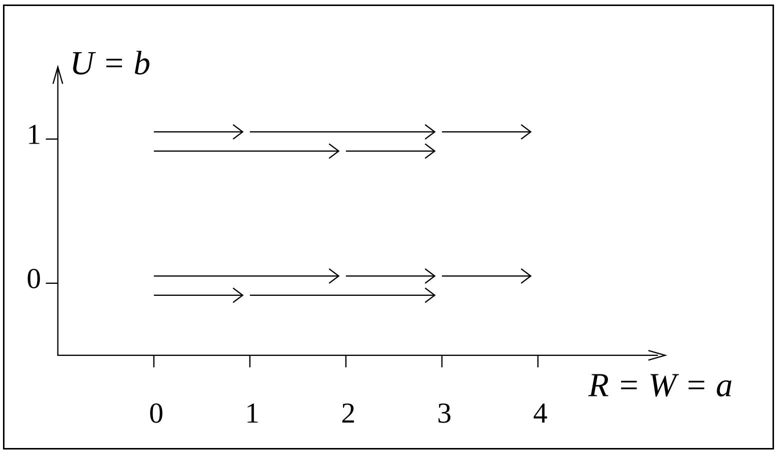
First note that it is not correct simply to use $\mathcal{A}(N)$ or $\mathcal{A}_i(N)$ in place of $M$ as was done in the definition of *writes*. The conditions

$$reads_{c(F,I,R,W,P,Q)}(N) \ \triangleq \ \Xi_R \mathbin{;} \mathcal{A}(N) \subseteq \mathcal{A}(N) \mathbin{;} \Xi_R$$

and

$$reads_{c(F,I,R,W,P,Q)}(N) \quad \triangleq \quad \forall i \in \mathbb{N}_1 \cdot \Xi_R ; \mathcal{A}_i(N) \subseteq \mathcal{A}_i(N) ; \Xi_R$$

are not strong enough as the example depicted in the following diagram shows.




Here we see that all the $\mathcal{A}_i(N)$ respect the read frame since for all $i$ we have

$$\{(a, b) \mid b = 0\} \lhd \mathcal{A}_i(N) \quad = \quad \{(a, b) \mid b = 1\} \lhd \mathcal{A}_i(N)$$

however, the overall behaviour does not respect $R$ since observing a particular sequence of values for $a$ would enable one to deduce the value of $b$. Thus, we need to consider each whole sequence in turn, rather than grouping the individual steps by taking the $\mathcal{A}_i$.

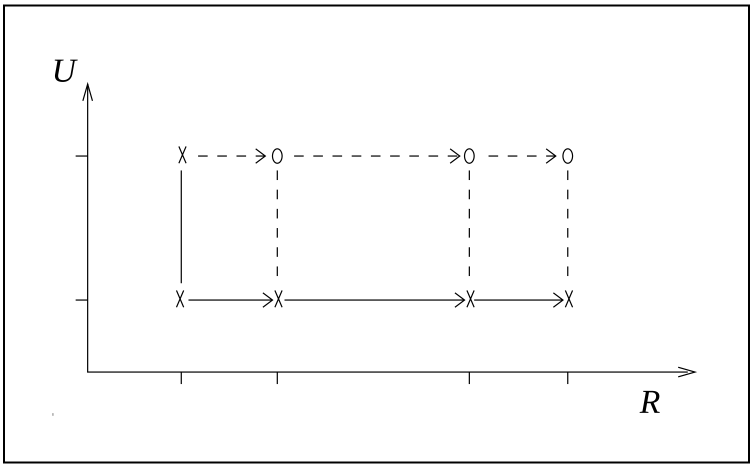
**Secure interpretation.** Following the approach in earlier chapters, we define a "secure" version of the non-reading condition which states that the value of the unread variables cannot be deduced by looking at the behaviour on the read variables

$$\forall \sigma_1^a \in N \cdot \forall \sigma \in \Sigma \cdot \sigma \; \Xi_R \; \sigma_1^a(1) \;\Rightarrow\; \exists \sigma_2^a \in N \cdot \sigma_2^a(1) = \sigma \wedge \sigma_2^a \Xi_R^a \sigma_1^a$$

where

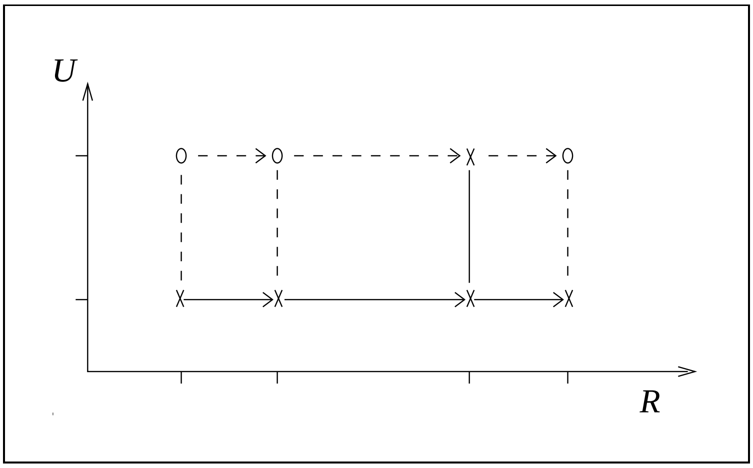
$$\sigma_1^a \Xi_R^a \sigma_2^a \quad \triangleq \quad \mathsf{inds}\, \sigma_1^a = \mathsf{inds}\, \sigma_2^a \wedge \forall i \in \mathsf{inds}\, \sigma_1^a \cdot \sigma_1^a(i) \; \Xi_R \; \sigma_2^a(i)$$

This is pictured in the following diagram where solid arrows and $\times$s depict "givens" and dashed arrows and 0s depict "existences".




One can immediately write an apparently stronger version of this criterion which is equivalent when $R \supseteq W$. This is done by generalising the position at which the alternative state is given

$$\forall \sigma_1^a \in N \cdot \forall \sigma \in \Sigma \cdot \forall i \in \mathbb{N}_1 \cdot \sigma \; \Xi_R \; \sigma_1^a(i) \;\Rightarrow\; \exists \sigma_2^a \in N \cdot \sigma_2^a(i) = \sigma \wedge \sigma_2^a \Xi_R^a \sigma_1^a$$




This condition gives a counterpart, with finer atomicity, to that given for the relational model in Chapter 5 and hence provides the basis for the persistent interpretation of the read and write frames and invariant, however, this model is still not rich enough to justify non-interference under interleaving of execution steps.

**No interleaving.** As discussed for the relational model in Chapter 4, the given conditions capture the *possibility* of read-respecting code rather than the *neccessity* of it. That is, if an interpretation satisfying the read constraint is a model of code at all, then it is a model of some code which respects the read frame. Conversely, when this condition does not hold, it is impossible to find a read-respecting implementation with this semantics.

However, as for the relational model, it is of course possible for code which does not respect the read frame to have this interpretation due to some 'unnecessary' access outside the read frame.

Herein lies the cause of the non-extensional behaviour of code under interference, and hence, the impossibility of a semantic criterion for non-interference under interleaving with this model. To see this, consider the following trivial example

$$\begin{aligned} \Sigma \ :: \ a &: \{0,1\} \\ b &: \{0,1\} \end{aligned}$$

*Op*

ext rd $a$ : $\{0,1\}$
    wr $a$ : $\{0,1\}$

pre $a = 0$

post $a = 1$

'Extensionally' the behaviour of this operation is extremely simple: $a$ will be 0 to start, may acquire any value during execution, but must be 1 at the end; $b$ will be unchanged throughout. Clearly, the interpretation satisfies the conditions given above and, indeed, the operation can be implemented by read-respecting code such as $a := 1$. But the prescribed extensional behaviour can also derive from more malevolent code which achieves exactly the same interpretation but does not respect the read frame and so is not so well behaved under interference. For example

if $b = 1$ then $a := b$ else $a := b + 1$

has the same interpretation as the simple assignment above, but now, if the value of $b$ is changed between the test and the assignment, then either $a = 0$ or $a = 2$ could result.

In general, even if we restrict our attention to cases which satisfy some semantic criterion such as the one given above, we cannot deduce the behaviour on resumption after an interruption simply from the behaviours without interruption. The existence of another execution sequence that 'mirrors' the uninterrupted one is not sufficient to ensure that the correspondence between the two behaviours reflects the fact that the two executions are taking the same route through some read-respecting code. It is equally possible that they are taking different routes through some code which does not respect the read frame and so one cannot predict the behaviour after a change in the supposedly unread variable. Thus it is impossible to give a semantics for interleaving based on an extensional model such as this and so one cannot give a semantic criterion for non-interference under interleaving.

## 7.4 Manipulating the overall frame

The semantics presented thus far has always assumed a fixed overall frame, however, the proof theory in Chapter 3 included the possibility of expanding and contracting it. Now we extend the semantic framework to model the manipulation of the variables in scope.

### 7.4.1 'Parallel' decomposition

For operations with overall frames that are completely disjoint, we can define a pseudo-parallel operator based on simultaneous, non-interfering execution. Its interpretation is simply

$$[\![Op_1 \parallel Op_2]\!] \quad \triangleq \quad [\![Op_1]\!] \times [\![Op_2]\!]$$

Here Cartesian product is lifted to the semantic domain in the obvious manner

$$\mathcal{S}_1 \times \mathcal{S}_2 \quad \triangleq \quad \{(T_1 \times T_2, M_1 \times M_2) \mid (T_i, M_i) \in \mathcal{S}_i\}$$

where Cartesian product on termination sets is the usual one, but Cartesian product for meaning relations is defined by taking all maplets of pairs rather than pairs of maplets

$$T_1 \times T_2 \quad \triangleq \quad \{(\sigma_1, \sigma_2) \mid \sigma_i \in T_i\}$$

$$M_1 \times M_2 \quad \triangleq \quad \{(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}) \mapsto (\sigma_1, \sigma_2) \mid (\overleftarrow{\sigma_i}, \sigma_i) \in M_i\}$$

The following lemma is proved in Appendix A.4.8

**Lemma**

For $i = 1, 2$, given

$$Op_i = mk\text{-}OpDef(F_i, I_i, R_i, W_i, P_i, Q_i)$$

such that $\mathsf{dom}\, F_2 \cap \mathsf{dom}\, F_1 = \{\,\}$, let

$$Op = mk\text{-}OpDef(F_1 \cup F_2, I_1 \wedge I_2, R_1 \cup R_2, W_1 \cup W_2, P_1 \wedge P_2, Q_1 \wedge Q_2)$$

then we have

(1) $\quad drwr_{Op_1}(T_1, M_1) \wedge drwr_{Op_2}(T_2, M_2) \;\Rightarrow\; drwr_{Op}(T_1 \times T_2, M_1 \times M_2)$

Now (1) immediately gives

$$\begin{aligned} &\big\{(T_1 \times T_2, M_1 \times M_2) \;\big|\; drwr_{Op_1}(T_1, M_1) \wedge drwr_{Op_2}(T_2, M_2)\big\} \\ &\qquad \subseteq \big\{(T_1 \times T_2, M_1 \times M_2) \;\big|\; drwr_{Op}(T_1 \times T_2, M_1 \times M_2)\big\} \end{aligned}$$

so

$$[\![Op_1 \parallel Op_2]\!] \subseteq [\![Op]\!]$$

and hence

$$Op \sqsubseteq Op_1 \parallel Op_2$$

This result tells us that if the frames and predicates of an implicit operation can be partitioned to give two disjoint, well-defined operation definitions, then we can develop these two parts separately. This is clearly a generalisation of the contract frame rule from Chapter 3

$$\frac{F_A \overset{ind}{\supseteq} F_C,\ (F_C, I_{F_C}, R, W, P, Q)\colon OpDef}{(F_A, I, R, W, P, Q) \sqsubseteq (F_C, I_{F_C}, R, W, P, Q)}$$

where $Op_2$, which is the identity on $F_A - F_C$, was left implicit. Note the requirement for indepenence of $F_C$ which ensures that the invariant can be written as the conjunction of $I_{F_C}$ and $I_{F_A - F_C}$.

### 7.4.2 Projection

As a counterpart to 'parallel' composition, we also extend the definition of projection to the semantic domain as follows

$$\mathcal{S}\big|_{F_1} = \{(T\big|_{F_1}, M\big|_{F_1}) \mid (T, M) \in \mathcal{S}\}$$

where

$$T\big|_{F_1} = \{\sigma\big|_{F_1} \mid \sigma \in T\} \quad \text{and} \quad M\big|_{F_1} = \{(\overleftarrow{\sigma}\big|_{F_1}, \sigma\big|_{F_1}) \mid (\overleftarrow{\sigma}, \sigma) \in M\}$$

Recall that the rule for expanding the overall frame uses a variable declaration of the form

$$\mathsf{var}\ \mathsf{F}_2\ \mathsf{in}\ \mathit{mk\text{-}OpDef}(F_1 \cup F_2, I_1, R_1 \cup S, W_1 \cup S, P_1, Q_1)\ \mathsf{end}$$

where $F_2$ is a declaration of new variables and $S = \mathsf{dom}\, F_2$

Now the fundamental property of the declaration of local variables is that they should not affect the (extensional) semantics of the operation. Thus we define[7] the semantics of variable declaration to be the projection onto the old variables

$$[\![\mathsf{var}\ F_2\ \mathsf{in}\ Op\ \mathsf{end}]\!] \quad \triangleq \quad [\![Op]\!]\big|_{F - F_2}$$

where $F$ is the overall frame of $Op$.

We will show below that, for the operations of the above lemma, provided $I_2 = P_2 = \mathsf{true}$, we have

[7] Note that this definition is not the same as requiring that $[\![Op]\!] = [\![Op_1]\!] \times [\![Op_2]\!]$ for some $Op_2$, since that would require $[\![Op]\!]$ to be constructible as the parallel composition of two separate operations which is not necessarily the case.

$$[\![Op]\!]\Big|_{F_1} = [\![Op_1]\!]$$

Since $R_2$, $W_2$, $Q_2$ are arbitrary, this is clearly stronger than the rule in Chapter 3 which requires that all the 'new' variables become available for both reading and writing, whilst all the 'new' predicates are trivial

$$\frac{\mathsf{dom}\, F \cap S = \{\,\}}{(F, I, R, W, P, Q) \sqsubseteq \mathsf{var}\ \mathsf{F}_2\ \mathsf{in}\ (F \cup S, I, R \cup S, W \cup S, P, Q)\ \mathsf{end}}$$

To prove the equality we will use the following result which is also proved in Appendix A.4.8.

**Lemma (cont.)**

For $Op_1$, $Op_2$ and $Op$ as above, if $I_2 = P_2 = \mathsf{true}$, we have

(2) $\quad drwr_{Op}(T, M) \;\Rightarrow\; drwr_{Op_1}(T\big|_{F_1}, M\big|_{F_1})$

Now (2) gives

$$\left\{(T\big|_{F_1}, M\big|_{F_1}) \;\middle|\; drwr_{Op}(T, M)\right\} \subseteq \left\{(T\big|_{F_1}, M\big|_{F_1}) \;\middle|\; drwr_{Op_1}(T\big|_{F_1}, M\big|_{F_1})\right\}$$

and so

$$[\![Op]\!]\Big|_{F_1} \subseteq [\![Op_1]\!]$$

Conversely, since projection is monotonic in containment, by (1) we have

$$[\![Op_1 \parallel Op_2]\!]\Big|_{F_1} \subseteq [\![Op]\!]\Big|_{F_1}$$

hence

$$[\![Op_1]\!] \subseteq [\![Op]\!]\Big|_{F_1}$$

as required.

The constraint required in the proof, that $P_2 = I_2 = \mathsf{true}$, can be understood as ensuring that no new, non-terminating behaviour is introduced by the declaration of the local variables. On the other hand, the freedom in $Q_2$ reflects the fact that an implementation can leave the local variables in any state it wishes.

The freedom in $R_2$ and $W_2$ gives rise to four extreme cases. The rule from Chapter 3 uses $R_2 = W_2 = \mathsf{dom}\, F_2$ which yields the usual semantics for declaration of variables, whilst the case where both frames are empty can be thought of as giving an interpretation to the 'closed-world' nature of code which does not change variables that are not in scope. The third case, where the local variables are read-only, could be regarded as corresponding to the declaration of constants, but perhaps the most interesting case is where the local variables are all write-only. In this case, the read frame constraint requires that the original values of the new variables can have no effect on the behaviour of the implementation. This corresponds to the syntactic condition that local variables are initialised before use and thus gives a semantic basis for what is usually regarded as a sound programming discipline.

# Chapter 8

# Conclusions

> *"It is indeed rather astonishing how little practical value scientific knowledge has for ordinary men,... We live either by rule of thumb or on other people's professional knowledge."*

It is perhaps not surprising that no single predominant theory has to date arisen as a basis for the practice of the engineering of software. Programs in electronic systems are as varied in form and as diverse in application as wheels in mechanical systems. Although there is an established foundational theory of recursive functions, this is at such a high level of abstraction that it is of little use in the practical design tasks facing the software engineer. As yet, the bridge between theory and practice amounts to no more than a number of disconnected stepping-stones: while each provides some assistance, no clear best path is established. Indeed, it sometimes seems that there are a greater number of formal techniques than there are examples of their use.

The explicit presentation of the read and write frames of operations is established practice in VDM. Their use can quickly convey an intended non-interference between operations, whilst abstraction is maintained through non-determinism which would otherwise have to be resolved before the non-interference could be demonstrated. In this way, it is possible to give a characterisation of a class of implementations which could not otherwise be specified, and hence a more abstract specification of an algorithm, freed from any "implementation bias" that would otherwise present in the order of steps. The naturalness of this interpretation of the frames is demonstrated by the "obvious" rules in the proof theory of Chapter 3, however, we have seen that existing semantic models do not provide a basis for this interpretation.

The major part of this work, Chapters 4, 5 and 6, has been directed towards the development of a semantic model to underpin the syntactic techniques proposed in Chapter 3. It has come as a considerable surprise, to the author at least, that the formal interpretation of such an innocuous part of an established notation, should require such a fundamental change to the received semantic model.

One can see the progression of this work as following the Hegelian dialectic of "thesis, antithesis, synthesis". Here, the 'thesis' put forward in the proof theory of Chapter 3, was countered by the 'antithesis' of negative result in Chapter 4, which lead to the 'synthesis' in Chapters 5, 6 and 7. It is interesting to note that it was at the point of synthesis, that the correspondence between read frames and bisimulation was observed which is perhaps the single most interesting result in the present work.

Naturally, as with any piece of work, this dissertation leaves many questions unanswered. We now conclude by briefly discussing some "matters arising" that would be worthy of future investigation.

**Interleaving.** In Section 7.2, we resolved an incompleteness in the proof theory by modelling intermediate states and so removed some unwanted refinements from the semantic model. Although this model preserves the non-interference result of Chapters 5 and 6, one's understanding of the properties of read and write respecting code might have suggested a stronger result involving the interleaving of execution steps. However, this is not a property of the present model. In order to recover non-interference under interleaving a further enrichement is required which in some way records the 'program counter' at the point of interruption.

In order to model shared-state parallelism in general, since it is not known from which state execution will be resumed after an interruption, one needs to describe the behaviour of the resuming operation from any state. In models based on resumptions (e.g. [Jon91]), the semantic domain not only encodes the behaviour of code through its "derivation tree", but also stipulates at each step the behaviour on resumption from any state. However, to model interleaved execution where frames are disjoint, a simpler model should suffice. For this case only the derivation tree should be required as the disjoint frames ensure that any interference affects only the unread variables and so does not alter any subsequent behaviour. It should then be possible to constrain the set of models to those which satisfy a 'no read' condition similar to that given for the simpler models, but which additionally ensures that the corresponding behaviours also correspond in the execution path taken. Such a model would then record not only the sequence of states visited, but would also label each step with information reflecting the path taken.

When non-interfering, the read and write frames, can be seen as a special case of the rely-guarantee conditions of [Jon81, Jon83a, Jon83b]. The read constraint states a reliance on certain variables not being changed by the environment, whilst the write frame asserts a guarantee that certain others will not be changed by the operation. It is the static nature of the frames that allows their modelling without the complexity of resumptions, however it might be interesting to carry the present work across to the more general setting.

**The spec – code continuum.** Although set in a context of the incremental development of operations, where implicit specifications and explicit pieces of code coexist during development, this thesis has solely considered the extension of *implicit* operation definitions with read and write frames, under the assumption that the read and write accesses made by code are syntactically obvious. It has, however, been noted several times that code can behave as if it respects the read and write frames, even though syntactically it does not. It would be interesting to consider the possibilities that might arise if code fragments were also annotated with read and write frames.

For example, $\{y\}, \{x\}: x := y$, would be the straightforward code reading $y$, writing $x$, which assigns the value of $y$ to $x$, whereas, $\{x\}, \{x\}: x := y$, would be required to achieve the same effect without reading $y$. This requirement might well be unimplementable in general but in a particular case, where perhaps the value of $y$ is known from an assertion, a read respecting coding could be devised.

Going one step further, even expressions could be annotated with their permitted read accesses. Thus $\{\}: y - 1$ would normally be undefined, whereas $\{\}: y - y$ could be interpreted as equivalent to $\{\}: 0$. With such extensions it should be possible to reason about non-interference at whatever level of granularity is required. It would be interesting to consider whether such extensions to the syntactic domain would yield any convenient programming forms.

**The interpretation of write-only variables.** We have studied at some length a number of the different models for the read frame which have differed only in the interpretation of variables that are writable but not readable. The must-write and may-write interpretations of write-only variables, have a direct correspondence to syntactic conditions applicable to code, but do not yield an orthogonal treatment of the read and write frames. The secure interpretation, on the other hand, does give orthogonality, but does not model a simple syntactic condition. Perhaps there is some useful "synthesis" to be sought here?

Although we have used the term 'secure' to describe the interpretation of write-only variables, no serious attempt has been made to compare this work to that in the security domain. The non-preservation of read frames by refinement can be seen as a particular case of a general result that security properties are not preserved by refinement (e.g. [Jac92]). Can anything be learned by applying the approach taken here, namely the encoding of refinements into the operation semantics, to the security problem?

**"Unification" with process algebra.** As stated above, perhaps the most far-reaching result in the present work is the correspondence between non-reading and bisimulation described in Section 5.3. This supports the interest raised in Section 2.4 in the dualism between model-oriented and process-based specifications. The ramifications of this correspondence have not been fully explored, but here we indicate some possibilities in this direction.

Firstly, the process algebraic view could be considered as a semantic model for the model-oriented specification. The algebraic laws of the semantic domain might then be used to explore alternative designs. An example of such a treatment is [Jon92], where a $\pi$-calculus [Mil91] semantics is given for a model-oriented specification language with dynamic, object-based, state manipulations. For VDM specifications, where state variables are declared statically, one would imagine that a CCS [Mil89] based semantics would suffice.

Another possibility has become evident in the MaFMeth project [BD94]. This concerns the evaluation of a methodology involving the derivation of test cases by the static analysis [DF93] of the predicates of an abstract model-oriented specification, and the subsequent use of these tests to validate the eventual implementation. Perhaps, by utilising the read frame we could restrict attention to those implementations which respect an equivalence class on the state space and thus we should be able to greatly reduce the number of test cases required. Furthermore, in order to utilise such sets of test cases, it is also necessary to define a series of operation calls that will take the system through the state changes required to exercise the tests – preferably in an efficient manner. Here, we wish to reason about sequences of operations, for which a process-based view of the abstract specification is more suited. In validating such process-based specifications, it is desirable to use equivalences to reduce the effective number of states for model checking. Could the equivalences introduced by the use of read frames in the model-oriented specification lead to a useful partitioning of the state space in this context?

In general, since the verification of read and write accesses of code is entirely mechanistic, any leverage that can be gained on the development task by their exploitation will be beneficial, replacing intricate reasoning with automatable checks.

# Acknowledgement

I would like to acknowledge the considerable contribution of my colleagues to this work. I am grateful to my supervisor Professor Cliff Jones for providing the framework upon which this thesis is based and for valuable guidance despite the geographical distance between Manchester and the Rutherford Appleton Laboratory. I am also grateful to Professor David Duce who has provided me with local support, particularly concerning administrative matters at RAL, and to Joe Morris and Bill Mitchell for their helpful comments on an earlier draft.

I would also like to thank my colleagues at RAL, in particular the two Brians, Ritchie and Matthews, Chris Reade and Mike Spivey who, together and separately, have helped to clarify these ideas as they are represented both in my mind and on the page.

# Bibliography


[Abr95] J-R. Abrial. *The B-Book*. Cambridge University Press, 1995. To appear.

[AK89] J.A. Ah-Kee. *Operation Decomposition Proof Obligations for Blocks and Procedures*. PhD thesis, University of Manchester, 1989.

[AL90] M. Adabi and L. Lamport. Composing Specifications. Technical Report 66, Digital Systems Research Centre, 1990.

[ALNS91] J.R. Abrial, M.K.O. Lee, D.S. Neilson, and P.N. Scharbach. The B method. In S. Prehn and W.J. Toetenel, editors, *Proceedings of VDM '91, Formal Software Development Methods*, volume 552 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[BA91] G. Bruns and S. Anderson. The formalization of a communications protocol. LFCS ECS-LFCS-91-137, University of Edinburgh, 1991.

[BA92a] G. Bruns and S. Anderson. The formalization of a communications protocol. LFCS/Adelard, University of Edinburgh, 1992.

[BA92b] G. Bruns and S. Anderson. The modal mu calculus. LFCS, University of Edinburgh, 1992.

[BA94] G. Bruns and S. Anderson. The formalization and analysis of a communications protocol. *Formal Aspects of Computing*, 6(1):92–112, 1994.

[Bac88] R.J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[BD94] J.C. Bicarregui and J. Dick. Mafmeth: The application measurement and dissemination of a mathematically formal methodology. *ERCIM News*, (18), July 1994. European Research Consortium in Informatics and Mathematics, Domaine de Voluceau, Rocquencourt, B.P. 105 F-78153 Le Chesnay Cedex, France, Article is available on World Wide Web, URL http://web.inf.rl.ac.uk/proj/mafmeth.html.

[BFL+94] J.C. Bicarregui, J.F. Fitzgerald, P.A. Lindsay, R. Moore, and B. Ritchie. *Proof in VDM – A Practitioner's Guide*. Springer-Verlag, 1994.

[Bic93] J.C. Bicarregui. Algorithm refinement with read and write frames. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Bic94] J.C. Bicarregui. Operation semantics with read and write frames. In *Sixth Refinement Workshop*, Workshops in Computer Science. Springer-Verlag, 1994.

[BL76] D.E. Bell and L.J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997, Mitre Corp., Bedford Mass., HQ Electron.Syst.Dov., Hanscom AFB, MA, TEch. Rep. ESD-TR-75-306 June 1975, 1976. Available as NTIS AD A023 588.

[BR89a] J.C. Bicarregui and B. Ritchie. The Mural proof obligation generation specification. Alvey Deliverable, IPSE 2.5 Project Document 060/00149.2.1, Rutherford Appleton Laboratory, 1989.

[BR89b] J.C. Bicarregui and B. Ritchie. Providing support for the formal development of software. In W. Schafer N. Madhavji and H. Weber, editors, *Proceedings of the First International Conference on Systems Development Environments and Factories*. Pitman, 1989.

[BR91] J.C. Bicarregui and B. Ritchie. Reasoning about VDM developments using the VDM support tool in Mural. In S.Prehn and W.J.Toetenel, editors, *Proceedings of VDM '91, Formal Software Development Methods*, volume 552 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[BR93] J.C. Bicarregui and B. Ritchie. Invariants, frames and postconditions: a comparison of the VDM and B notations. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[BvW90] R.J.R. Back and J. von Wright. Duality in specification languages: A lattice-theoretical approach. *Acta Informatica*, 27:583–625, 1990.

[DF93] Jeremy Dick and Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[Fit91] J.S. Fitzgerald. *Modularity in Model-Oriented Formal Specifications and its Interaction with Formal Reasoning*. PhD thesis, University of Manchester, 1991. Also available as Technical Report UMCS-91-11-2.

[GP92] C. George and S. Prehn. The Raise justification handbook. Technical Report LACOS/CRI/DOC/7, CRI, Compter Resources International A/S., Bregnerodvej 144, DK-3460 Birkerod, Denmark., 1992.

[Har40] G.H. Hardy. *A Mathematician's Apology.* Cambridge University Press, 1940. Canto edition 1992, p. 85.

[Hay93] I Hayes. *Specification Case Studies.* Prentice Hall, 1993. Second edition.

[HHJ+85] C.A.R. Hoare, I.J. Hayes, He J.F., C.C. Morgan, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Sufrin. Data refinement refined, 1985. Unpublished.

[HM85] M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *JACM 32(1)*, 1985.

[ISO93] Information Technology Programming Languages – VDM-SL. Technical report, First Committee Draft Standard: CD 13817-1, November 1993. ISO/IEC JTC1/SC22/WG19 N-20.

[Jac92] Jeremy Jacob. Basic Theorems about Security. *Journal of Computer Security*, 1(4):385–411, 1992.

[Jon80] C.B. Jones. *Software Development a Rigorous Approach.* Prentice Hall, 1980.

[Jon81] C.B. Jones. *Development Methods for Computer Programmes Including a Notion of Interference.* PhD thesis, Oxford Univ., Comp. Lab., June 1981. PRG Monograph 25.

[Jon83a] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Manson, editor, *Proceedings of IFIP '83*, pages 321–332. IFIP, North-Holland, 1983.

[Jon83b] C.B. Jones. Tentative steps towards a development method for interfering programs. *Transactions on Programming Languages and Systems*, 5(4):596–619, Oct. 1983.

[Jon87] C.B. Jones. VDM proof obligations and their justification. In M. Mac an Airchinnigh D. Bjørner, C.B. Jones and E.J. Neuhold, editors, *Proceedings of VDM '87*, volume 252 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987.

[Jon90a] C.B. Jones. *Systematic Software Devlopment using VDM*, chapter 10. Prentice Hall, 1990.

[Jon90b] C.B. Jones. *Systematic Software Devlopment using VDM.* Prentice Hall, second edition, 1990.

[Jon91] C.B. Jones. Interference resumed. Technical Report UMCS-91-5-1, University of Manchester, 1991.

[Jon92] C.B. Jones. An object based design method for concurrent programs. Technical Report UMCS-92-12-1, University of Manchester, 1992.

[Kin90] S. King. Z and the refinement calculus. In C.A.R. Hoare D. Bjørner and Hans Langmaack, editors, *Proceedings of VDM '90, VDM and Z*, volume 428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Lam91] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Centre, 1991.

[Mil88] R. Milne. Proof rules for VDM statements. In L. Marshall R. Bloomfield and R. Jones, editors, *Proceedings of VDM '88*, volume 328 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91] R. Milner. A polyadic $\pi$-calculus: A tutorial. Laboratory for Foundations of Computer Science ECS-LFCS-91-180, University of Edinburgh, 1991.

[Mor87] J. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9:287–306, 1987.

[Mor88a] C. Morgan. Data refinement by miracles. *Inf. Proc. Letters*, 26(5), Jan. 1988.

[Mor88b] C. Morgan. The specification statement. In *TOPLAS 10*, 1988.

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

[MRG88] C. Morgan, K. Robinson, and P. Gardiner. On the refinement calculus. Technical Monograph PRG-70, Oxford University, 1988.

[RLG92] The Raise Language Group. *The Raise Specification Language*. Prentice Hall, 1992.

[SBC+89] L.L. Santoline, M.D. Bowers, A.W. Crew, C.J. Roslund, and W.D. Grist III. Multiprocessor shared-memory information exchange. *IEEE Transactions on Nuclear Science*, 36.1:626–633, 1989.

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

[War93] N. Ward. Adding specification constructors to the refinement calculus. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Wit22] L. Wittgenstein. Tractatus logico-philosophicus. *Routledge and Kegan Paul*, First published 1922. Translation by C.K. Ogden.

# Appendix A

# Deferred Proofs

## A.1 Proofs from Chapter 3

### A.1.1 Criteria for independence of a part of the state (Section 3.2.3)

A part $S$ of composite type $F \dagger I$ is an independent part of $F$ if and only if there are predicates $I_S\colon Exp(S)$ and $I_T\colon Exp(F-S)$ such that

$$F \dagger I = F \dagger (I_S \wedge I_T)$$

**Proof** Define

$$I_S \triangleq \exists \mathsf{F}{-}\mathsf{S} \cdot I$$

We are required to prove

$$\begin{array}{rcl} S \overset{ind}{\subseteq} F & \triangleq & \forall \mathsf{F}, \overleftarrow{\mathsf{F}} \cdot I \wedge \overleftarrow{I} \;\Leftrightarrow\; \overleftarrow{I}^{S} \wedge \overleftarrow{I}^{T} \\ & \Leftrightarrow & \forall \mathsf{F}, \overleftarrow{\mathsf{F}} \cdot I \wedge \overleftarrow{I} \;\Rightarrow\; \overleftarrow{I}^{S} \\ & \Leftrightarrow & \forall \mathsf{F} \cdot I_S \wedge I_{F-S} \Leftrightarrow I \end{array}$$

**Proof**

1) $\Rightarrow$ 2) is trivial.

2) $\Rightarrow$ 3)

$$\begin{array}{rcl} I_S \wedge I_{F-S} & \triangleq & \exists \overleftarrow{\mathsf{S}} \cdot \overleftarrow{I}^{S} \wedge \exists \overleftarrow{\mathsf{F}-\mathsf{S}} \cdot \overleftarrow{I}^{F-S} \\ & \Rightarrow & \exists \overleftarrow{\mathsf{F}} \cdot \overleftarrow{I}^{S} \wedge \overleftarrow{I}^{F-S} \\ & \Rightarrow & I \qquad \text{by (2) (taking } S \text{ part of } \overleftarrow{I}^{F-S} \text{ and } F-S \text{ part of } \overleftarrow{I}^{S}) \end{array}$$

3) $\Rightarrow$ 1)

$$\begin{array}{llll} I \wedge \overleftarrow{I} & \Rightarrow & I_S \wedge I_{F-S} \wedge \overleftarrow{I_S} \wedge \overleftarrow{I_{F-S}} & \text{by (3)} \Leftarrow \\ & \Rightarrow & I_S \wedge \overleftarrow{I_{F-S}} \wedge \overleftarrow{I_S} \wedge I_{F-S} & \\ & \Rightarrow & \overleftarrow{I}^{F-S} \wedge \overleftarrow{I}^{S} & \text{by (3)} \Rightarrow \end{array}$$

## A.2 Proofs from Chapter 5

### A.2.1 Equivalent forms for readsT (Section 5.4)

**lemma**

For a relation $R{:}\,\mathcal{P}(\Sigma \times \Sigma)$ and a set $S{:}\,\mathcal{P}\Sigma$ we have

$$R;T \subseteq T;R \;\Leftrightarrow\; R\,(\!|\, T \,|\!) \subseteq T$$

and

$$R;T \subseteq T;R \;\Leftrightarrow\; R\,(\!|\, \Sigma - T \,|\!) \subseteq (\Sigma - T)$$

Furthermore, if $R$ is symmetric, then

$$R;T \subseteq T;R \;\Leftrightarrow\; R;T = T;R$$

and, if $R \supseteq Id_\Sigma$, then

$$R;T \subseteq T;R \;\Leftrightarrow\; R\,(\!|\, T \,|\!) = T$$

**proof**

1a) $\Rightarrow$

$$\begin{array}{llll} \sigma \in R\,(\!|\, T \,|\!) & \Rightarrow & \exists \overleftarrow{\sigma} \cdot \overleftarrow{\sigma} R \sigma \wedge \overleftarrow{\sigma} \in T & \\ & \Rightarrow & (\overleftarrow{\sigma}, \sigma) \in T;R & \\ & \Rightarrow & (\overleftarrow{\sigma}, \sigma) \in R;T & \Xi_R;T \subseteq T;\Xi_R \\ & \Rightarrow & \sigma \in T & \end{array}$$

1b) $\Leftarrow$

$$\begin{array}{llll} (\overleftarrow{\sigma}, \sigma) \in T;R & \Rightarrow & \sigma \in R\,(\!|\, T \,|\!) & \\ & \Rightarrow & \sigma \in T & R\,(\!|\, T \,|\!) \subseteq T \\ & \Rightarrow & (\overleftarrow{\sigma}, \sigma) \in R;T & \end{array}$$

2a) $\Rightarrow$

$$\begin{array}{rcll}
\sigma \in R\,(\!|\, \Sigma - T \,|\!) & \Rightarrow & \exists \overleftarrow{\sigma} \cdot \overleftarrow{\sigma}\, R \sigma \wedge \overleftarrow{\sigma} \notin T & \\
& \Rightarrow & \exists \overleftarrow{\sigma} \cdot (\overleftarrow{\sigma}, \sigma) \notin T;R & \\
& \Rightarrow & \exists \overleftarrow{\sigma} \cdot (\overleftarrow{\sigma}, \sigma) \notin R;T & \text{since } T;R \supseteq R;T \\
& \Rightarrow & \sigma \notin T & \text{since } \overleftarrow{\sigma}\, R \sigma
\end{array}$$

2b) $\Leftarrow$

$$\begin{array}{rcll}
(\overleftarrow{\sigma}, \sigma) \in T;R & \Rightarrow & \overleftarrow{\sigma} \in T \wedge \overleftarrow{\sigma}\, R\sigma & \\
& \Rightarrow & \overleftarrow{\sigma} \notin (\Sigma - T) \wedge \overleftarrow{\sigma}\, R\sigma & \\
& \Rightarrow & \overleftarrow{\sigma} \notin R(\Sigma - T) \wedge \overleftarrow{\sigma}\, R\sigma & \text{since } R(\Sigma - T) \subseteq T \\
& \Rightarrow & \sigma \notin (\Sigma - T) \wedge \overleftarrow{\sigma}\, R\sigma & \\
& \Rightarrow & \sigma \in T \wedge \overleftarrow{\sigma}\, R\sigma & \\
& \Rightarrow & (\overleftarrow{\sigma}, \sigma) \in R;T &
\end{array}$$

3a) $\Rightarrow$

$$\begin{array}{rcll}
(\overleftarrow{\sigma}, \sigma) \in R;T & \Rightarrow & \overleftarrow{\sigma}\, R\sigma \wedge \sigma \in T & \\
& \Rightarrow & \overleftarrow{\sigma} \in R\,(\!|\, T \,|\!) & R \text{ symmetric} \\
& \Rightarrow & \overleftarrow{\sigma} \in T & R\,(\!|\, T \,|\!) \subseteq T \\
& \Rightarrow & (\overleftarrow{\sigma}, \sigma) \in T;R &
\end{array}$$

3b) $\Leftarrow$, is trivial.

4a) $\Rightarrow$

$$\begin{array}{rcll}
T & = & Id_\Sigma\,(\!|\, T \,|\!) & \\
& \subseteq & R\,(\!|\, T \,|\!) & R \supseteq Id_\Sigma
\end{array}$$

4b) $\Leftarrow$, is trivial.

## A.3 Proofs from Chapter 6

### A.3.1 Unioning over fixpoints (Section 6.8)

**Lemma**

$$\bigcup_i \mathit{fix}(\lambda s \cdot \mathcal{G}_i(s)) = \mathit{fix}(\lambda s \cdot \bigcup_i \mathcal{G}_i(s))$$

and

$$\bigcap_i \mathit{fix}(\lambda s \cdot \mathcal{G}_i(s)) = \mathit{fix}(\lambda s \cdot \bigcap_i \mathcal{G}_i(s))$$

**Proof**

We use fixpoint induction

$$\frac{P(\bigcup_{n \geq 0} \mathcal{F}^n(\{\,\}))}{P(\mathit{fix}(\lambda s \cdot \mathcal{F}^n(s))}$$

as follows.

For $i \in I$ define

$$P_i(S) \quad \triangleq \quad \bigcup_{n \geq 0} \mathcal{F}_i^n = S$$

and

$$P'(S) \quad \triangleq \quad \bigcup_{i \in I} \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)) = S$$

Now

| | | |
|---|---|---|
| 1 | $\bigcup_{n \geq 0} \mathcal{F}_i^n(\{\,\}) = \bigcup_{n \geq 0} \mathcal{F}_i^n(\{\,\})$ | =-refl |
| 2 | $P_i(\bigcup_{n \geq 0} \mathcal{F}_i^n(\{\,\}))$ | $P$ |
| 3 | $P_i(\mathit{fix}(\lambda s \cdot \mathcal{F}_i(s))$ | *fix* |
| 4 | $\bigcup_{n \geq 0} \mathcal{F}_i^n(\{\,\}) = \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)$ | $P$ |
| 5 | $\bigcup_{i \in I} \bigcup_{n \geq 0} \mathcal{F}_i^n(\{\,\}) = \bigcup_{i \in I} \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)$ | substitutivity |
| 6 | $\bigcup_{n \geq 0} \bigcup_{i \in I} \mathcal{F}_i^n(\{\,\}) = \bigcup_{i \in I} \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)$ | $\cup$-comm |
| 7 | $P'(\bigcup_{n \geq 0} \bigcup_{i \in I} \mathcal{F}_i^n(\{\,\}))$ | $P'$ |
| 8 | $P'(\mathit{fix}(\lambda s \cdot \bigcup_{i \in I} \mathcal{F}_i(s))$ | *fix* |
| Infer | $\bigcup_{i \in I} \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)) = \mathit{fix}(\lambda s \cdot \bigcup_{i \in I} \mathcal{F}_i(s)$ | $P'$ |

The proof of the second equality follows similarly with

$$P'(S) \triangleq \bigcap_{i \in I} \mathit{fix}(\lambda s \cdot \mathcal{F}_i(s)) = S$$

### A.3.2 Intersecting frames distrubutes through operation definition (Section 6.8)

**Theorem**

$$[\![ \mathit{mk\text{-}OpDef}(F, I, R_\cap, W, P, Q) ]\!] = \bigcap_{i=1,2} [\![ \mathit{mk\text{-}OpDef}(F, I, R_i, W, P, Q) ]\!]$$

$$[\![ \mathit{mk\text{-}OpDef}(F, I, R, W_\cap, P, Q) ]\!] = \bigcap_{i=1,2} [\![ \mathit{mk\text{-}OpDef}(F, I, R, W_i, P, Q) ]\!]$$

**Proof** *writes*

We show $\mathit{writes}(W_\cap, M) \;\Leftrightarrow\; \forall i \cdot \mathit{writes}(W_i, M)$.

$$\begin{aligned}
\mathit{writes}(W_\cap, M) \quad &\Leftrightarrow \quad \forall (\overleftarrow{\sigma}, \sigma) \in M \cdot \overleftarrow{\sigma}\Big|_{F-W_\cap} = \sigma\Big|_{F-W_\cap} \\
&\Leftrightarrow \quad \forall (\overleftarrow{\sigma}, \sigma) \in M \cdot \forall i \cdot \overleftarrow{\sigma}\Big|_{F-W_i} = \sigma\Big|_{F-W_i} \\
&\Leftrightarrow \quad \forall i \cdot \forall (\overleftarrow{\sigma}, \sigma) \in M \cdot \overleftarrow{\sigma}\Big|_{F-W_i} = \sigma\Big|_{F-W_i} \\
&\Leftrightarrow \quad \forall i \cdot \mathit{writes}(W_i, M)
\end{aligned}$$

**Proof** *reads*

We show $\mathit{reads}(R_\cap, M) \;\Leftrightarrow\; \forall i \cdot \mathit{reads}(R_i, M)$.

$\Rightarrow$ is trivial since $\forall i \cdot R_\cap \subseteq R_i$.

To see $\Leftarrow$ we argue as follows.

Let $\overleftarrow{\sigma_1} \Xi_{R_1} \overleftarrow{\sigma_2} M \sigma_2$ and construct $\overleftarrow{\sigma_3} = (\overleftarrow{\sigma_1}\Big|_{F-R_1}, \overleftarrow{\sigma_2}\Big|_{R_1})$.

Then $\overleftarrow{\sigma_3} \Xi_{R_1} \overleftarrow{\sigma_2} M \sigma_2$, and so $\exists \sigma_3 \cdot \overleftarrow{\sigma_3} M \sigma_3 \Xi_{R_1} \sigma_2$.

Now $\overleftarrow{\sigma_1} \Xi_{R_2} \overleftarrow{\sigma_3} M \sigma_3$, so $\exists \sigma_1 \cdot \overleftarrow{\sigma_1} M \sigma_1 \Xi_{R_2} \sigma_3$.

Hence $\overleftarrow{\sigma_1} M \sigma_1 \Xi_{R_1} \sigma_2$ as required.

### A.3.3 Minor lemmas used in Chapter 6

**Lemma** $\Xi_S$ idempotent

$$\Xi_S ; \Xi_S = \Xi_S$$

**Proof**

$$
\begin{array}{rcll}
(\overleftarrow{\sigma}, \sigma) \in \Xi_S & \Leftrightarrow & \overleftarrow{\sigma}\big|_S = \sigma\big|_S & \text{defn} \\
& \Leftrightarrow & \exists \sigma' \cdot \overleftarrow{\sigma}\big|_S = \sigma'\big|_S = \sigma\big|_S & \text{choose for example } \sigma' = \sigma \\
& \Leftrightarrow & (\overleftarrow{\sigma}, \sigma) \in \Xi_S ; \Xi_S & \text{defn}
\end{array}
$$

**Lemma** domain of sequential composition

$\mathsf{dom}\,(R_1 ; R_2) = \mathsf{dom}\,(R_1 \rhd \mathsf{dom}\, R_2)$

**Proof**

$$
\begin{array}{rcl}
\sigma \in \mathsf{dom}\,(R_1 \rhd \mathsf{dom}\, R_2) & \Leftrightarrow & \exists \sigma_1 \colon \Sigma \cdot (\sigma, \sigma_1) \in R_1 \land \sigma_1 \in \mathsf{dom}\, R_2 \\
& \Leftrightarrow & \exists \sigma_1, \sigma_2 \colon \Sigma \cdot (\sigma, \sigma_1) \in R_1 \land (\sigma_1, \sigma_2) \in R_2 \\
& \Leftrightarrow & \exists \sigma_2 \colon \Sigma \cdot (\sigma, \sigma_2) \in (R_1 ; R_2) \\
& \Leftrightarrow & \sigma \in \mathsf{dom}\,(R_1 ; R_2)
\end{array}
$$

## A.3.4 Non-interference (Section 6.10).

We will prove the following result which immediately yields the non-interference theorem of Section 6.10.

$$
\frac{
\begin{array}{c}
Op_1 = mk\text{-}OpDef(F, I, R_1, W_1, P_1, Q_1) \\
Op_2 = mk\text{-}OpDef(F, I, R_2, W_2, P_2, Q_2) \\
F - W_2 \supseteq R_1 \supseteq W_1 \\
F - W_1 \supseteq R_2 \supseteq W_2 \\
Op_1 \sqsubseteq Op_1' \\
Op_2 \sqsubseteq Op_2'
\end{array}
}{
[\![ Op_1' ; Op_2' ]\!] = [\![ Op_2' ; Op_1' ]\!]
}
$$

**Proof**

Given two operations

$Op_i \quad \triangleq \quad mk\text{-}OpDef(F, I, R_i, W_i, P_i, Q_i), \quad i = 1, 2$

such that

$W_1 \subseteq F - R_2 \quad \text{and} \quad W_2 \subseteq F - R_1$

and provided

$R_i \supseteq W_i \quad i = 1, 2$

and given $Op'_1$ and $Op'_2$ such that

$$Op_i \sqsubseteq Op'_i \quad i = 1, 2$$

we are required to prove

$$[\![Op'_1 ; Op'_2]\!] = [\![Op'_2 ; Op'_1]\!]$$

By symmetry, it suffices to prove

$$\forall (T, M) \in [\![Op'_1 ; Op'_2]\!] \cdot (T, M) \in [\![Op'_2 ; Op'_1]\!]$$

that is

$$\begin{array}{l} \forall (T_1, M_1) \in [\![Op'_1]\!] \cdot \forall (T_2, M_2) \in [\![Op'_2]\!] \cdot \\ \quad \exists (T'_1, M'_1) \in [\![Op'_1]\!] \cdot \exists (T'_2, M'_2) \in [\![Op'_2]\!] \cdot \\ \qquad T_2 - \mathsf{dom}\,(M_2 \not\rhd T_1) = T'_1 - \mathsf{dom}\,(M'_1 \not\rhd T'_2) \wedge \\ \qquad M_2 ; M_1 = M'_1 ; M'_2 \end{array}$$

Now, this is ensured if we choose $M'_i = M_i$ and $T_i = T'_i$ since we will show that

$$\begin{array}{l} \forall (T_1, M_1) \in [\![Op'_1]\!] \cdot \forall (T_2, M_2) \in [\![Op'_2]\!] \cdot \\ \qquad T_2 - \mathsf{dom}\,(M_2 \not\rhd T_1) = T_1 - \mathsf{dom}\,(M_1 \not\rhd T_2) \wedge \\ \qquad M_2 ; M_1 = M_1 ; M_2 \end{array}$$

Firstly to show

$$M_2 ; M_1 = M_1 ; M_2$$

observe that

$$\begin{array}{llll} (\overleftarrow{\sigma_2}, \sigma_1) \in M_1 ; M_2 & \Rightarrow & \exists \overleftarrow{\sigma_1} \cdot \overleftarrow{\sigma_2}\ M_1\ \overleftarrow{\sigma_1}\ M_2\ \sigma_1 & \text{def'n} \\ & \Rightarrow & \exists \overleftarrow{\sigma_1} \cdot \overleftarrow{\sigma_2}\ \Xi_{F-W_1}\ \overleftarrow{\sigma_1}\ M_2\ \sigma_1 & \mathit{writes}(W_1, M_1) \\ & \Rightarrow & \exists \overleftarrow{\sigma_1} \cdot \overleftarrow{\sigma_2}\ \Xi_{R_2}\ \overleftarrow{\sigma_1}\ M_2\ \sigma_1 & R_2 \subseteq F - W_1 \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \overleftarrow{\sigma_2}\ M_2\ \sigma_2\ \Xi_{R_2}\ \sigma_1 & \mathit{reads}(R_2, M_2) \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \overleftarrow{\sigma_2}\ \Xi_{F-W_2}\ \sigma_2 & \mathit{writes}(W_2, M_2) \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \overleftarrow{\sigma_2}\ \Xi_{R_1}\ \sigma_2 & R_1 \subseteq F - W_2 \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \sigma_2\ \Xi_{R_1}\ \overleftarrow{\sigma_2} & R_1 \text{ Symmetric} \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \sigma_2\ \Xi_{R_1}\ \overleftarrow{\sigma_2}\ M_1\ \overleftarrow{\sigma_1} & \text{and intro} \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2, \sigma'_1 \cdot \sigma_2\ M_1\ \sigma'_1\ \Xi_{R_1}\ \overleftarrow{\sigma_1} & \mathit{reads}(R_1, M_1) \\ & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2, \sigma'_1 \cdot \overleftarrow{\sigma_2}\ M_2\ \sigma_2\ M_1\ \sigma'_1 & \text{and elim/intro} \\ & \Rightarrow & (\overleftarrow{\sigma_2}, \sigma'_1) \in M_2 ; M_1 & \text{def'n} \end{array}$$

But $\sigma'_1 = \sigma_1$ since

$$\sigma'_1\ \Xi_{R_1}\ \overleftarrow{\sigma_1}\ \Xi_{F-W_2}\ \sigma_1 \text{ and } \sigma'_1\ \Xi_{F-W_1}\ \sigma_2\ \Xi_{R_2}\ \sigma_1$$

so

$$\sigma_1' \ \Xi_{R_1 \cup R_2} \ \sigma_1$$

Furthermore, we have

$$\sigma_1' \ \Xi_{F-(W_1 \cup W_2)} \ \sigma_1$$

thus we have the result since we have assumed $R_i \supseteq W_i$.

Now to show

$$T_2 - \mathsf{dom}\,(M_2 \mathrel{\ntriangleright} T_1) = T_1 - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2)$$

First note that

$$\sigma \in T_1 - \mathsf{dom}\,(M_1 \mathrel{\ntriangleright} T_2) \;\Leftrightarrow\; \sigma \in T_1 \wedge M_1 (\!| \{\sigma\} |\!) \subseteq T_2$$

So we will show

$$\sigma \in T_1 \wedge M_1 (\!| \{\sigma\} |\!) \subseteq T_2 \;\Rightarrow\; \sigma \in T_2 \wedge M_2 (\!| \{\sigma\} |\!) \subseteq T_1$$

Now for $\sigma \in T_1$ we have

$$\begin{array}{rcll} M_2 (\!| \{\sigma\} |\!) & \subseteq & \Xi_{R_1} (\!| \{\sigma\} |\!) & \text{since } M_2 \subseteq \Xi_{F-W_2} \subseteq \Xi_{R_1} \\ & \subseteq & T_1 & \text{since } \Xi_{R_1} (\!| T_1 |\!) \subseteq T_1 \end{array}$$

also, $M_1 (\!| \{\sigma\} |\!) \subseteq T_2$ gives

$$\exists \sigma' \cdot \sigma M_1 \sigma' \wedge \sigma' \in T_2$$

since $\sigma \in T_1$ and $T_1 \subseteq \mathsf{dom}\, M_1$. But $M_1 \subseteq \Xi_{F-W_1} \subseteq \Xi_{R_2}$, so

$$\exists \sigma' \cdot \sigma \Xi_{R_2} \sigma' \wedge \sigma' \in T_2$$

so

$$\sigma \in \Xi_{R_2} (\!| T_2 |\!) \subseteq T_2$$

as required

## A.4 Proofs from Chapter 7

### A.4.1 Secure semantics - no intro write-only (Section 7.2.4)

#### Example

For the following abstract and concrete operations

| | |
|---|---|
| $Op_a$ | $Op_c$ |
| ext rd $x \quad : \mathbb{N}$ | ext rd $\{\}$ |
| wr $x, y : \mathbb{N}$ | wr $x, y : \mathbb{N}$ |
| post $x = \overleftarrow{y} \wedge y = 0$ | post $x = \overleftarrow{y} \wedge y = 0$ |

we show below that although the only difference between the two operations is that the latter has a smaller read frame, there is an $M$ in $[\![Op_c]\!]$ which is not in $[\![Op_a]\!]$ and thus $Op_a \not\sqsubseteq Op_c$.

To see this, note

$$\begin{array}{lcll}
\Xi_{R_a} & \triangleq & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (\overleftarrow{x}, y) & \mid \overleftarrow{x}, \overleftarrow{y}, y \colon \{0,1\}\} \\
\Xi_{R_c} & \triangleq & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (x, y) & \mid \overleftarrow{x}, \overleftarrow{y}, x, y \colon \{0,1\} \} \\
M_0 & \triangleq & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (\overleftarrow{y}, 0) & \mid \overleftarrow{x}, \overleftarrow{y} \colon \{0,1\}\}
\end{array}$$

So we do not have $reads(R_a, M_0)$ for the abstract operation since $\overleftarrow{y}$ affects $x$ which is read, i.e.

$$\begin{array}{lcll}
\Xi_{R_a};M_0 & = & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (x, 0) & \mid \overleftarrow{x}, \overleftarrow{y}, x \colon \{0,1\} \} \\
\not\subseteq & & & \\
M_0;\Xi_{R_a} & = & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (\overleftarrow{y}, y) & \mid \overleftarrow{x}, \overleftarrow{y}, y \colon \{0,1\} \}
\end{array}$$

Thus $M_0 \notin [\![Op_a]\!]$.

However, we do have $reads(R_c, M_0)$ since $\overleftarrow{y}$ only effects $x$ which is not read by $Op_c$

$$\begin{array}{lcll}
\Xi_{R_c};M_0 & = & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (x, 0) & \mid \overleftarrow{x}, \overleftarrow{y}, x \colon \{0,1\} \} \\
\subseteq & & & \\
M_0;\Xi_{R_c} & = & \{ (\overleftarrow{x}, \overleftarrow{y}) \mapsto (x, y) & \mid \overleftarrow{x}, \overleftarrow{y}, x, y \colon \{0,1\} \}
\end{array}$$

So $M_0 \in [\![Op_c]\!]$.

Thus $Op_a$ is unimplementable whereas $Op_c$ is implementable.

## A.4.2 Secure2 is too weak (Section 7.2.4)

### Lemma

$reads_{secure_2}$ is weaker than $reads_{secure}$ when $R \supseteq W$, ie.

$$\frac{\begin{array}{c} R \supseteq W \\ writes(W, M) \\ reads_{secure}(R, M) \end{array}}{reads_{secure_2}(R, M)}$$

**Proof**

w.l.o.g. let $\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1$ such that $\overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1 \wedge \overleftarrow{\sigma_1} \Xi_R \overleftarrow{\sigma_2}$ be given (since if either condition does not hold we recover $reads_{secure_2}$ trivially).

Now by $reads_{secure}$, choose $\sigma_2$ so that $\overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2 \wedge \sigma_1 \Xi_R \sigma_2$

Now given any $x \in F - R$ such that $\overleftarrow{\sigma_1} \Xi_{F-\{x\}} \overleftarrow{\sigma_2}$ we have $\sigma_1 \Xi_R \sigma_2$ by $reads_{secure}$. Furthermore, since $F - \{x\} \supseteq R$ we have

$$\sigma_1 \Xi_{F-R} \overleftarrow{\sigma_1} \Xi_{F-\{x\}} \overleftarrow{\sigma_2} \Xi_{F-R} \sigma_2$$

since $F - R \subseteq F - W$ and $writes(W, M)$. So we also have

$$\sigma_1 \Xi_{F-R-\{x\}} \sigma_2$$

as required.

**Example**

$reads_{secure_2}$ is too weak.

Consider

$$\begin{aligned} \Sigma \ :: \ & a: \{0,1\} \\ & b: \{0,1\} \\ & c: \{0,1\} \end{aligned}$$

$$inv\text{-}\Sigma(a, b, c) \quad \triangleq \quad a = b$$

$Op$
ext rd $c$:$\{0, 1\}$
  wr $c$:$\{0, 1\}$
pre true
post $2 * c + a + b = 2$

This clearly breaks the read constraint as we cannot choose an appropriate $c$ without knowledge of $a$ or $b$.

But $reads_{secure_2}$ does holds since

$$\forall x \in F - R \cdot \overleftarrow{\sigma_1} \Xi_{F-\{x\}} \overleftarrow{\sigma_2} \ \Rightarrow \ \overleftarrow{\sigma_1} = \overleftarrow{\sigma_2}$$

For example $\overleftarrow{\sigma_1} \Xi_{\{a\}} \overleftarrow{\sigma_2} \ \Rightarrow \ \sigma_1\Big|_{\{b\}} = \overleftarrow{\sigma_1}\Big|_{\{a\}} = \overleftarrow{\sigma_2}\Big|_{\{a\}} = \sigma_2\Big|_{\{b\}}$.

(Note that $reads_{secure}$ does not hold, for choosing $\overleftarrow{\sigma_1} = (0, 0, 0), \overleftarrow{\sigma_2} = (0, 0, 1), \overleftarrow{\sigma_1} = (1, 1, 0)$ gives $\overleftarrow{\sigma_1} \Xi_{\{c\}} \overleftarrow{\sigma_2} \wedge \overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1$. But there does not exists a suitable $\sigma_2$ since $(1, 1, 0) \mapsto (1, 1, 1) \notin [\![Op]\!]$ and $(1, 1, 0) \mapsto (0, 0, 1) \notin [\![Op]\!]$.)

### A.4.3 Secures equivalent (Section 7.2.5)

**Lemma**

$$reads_{secure_3} \ \Leftrightarrow \ reads_{secure'_3} \ \Leftrightarrow \ reads_{secure''_3}$$

**Proof**

(a)

$$reads_{secure_3} \;\Rightarrow\; reads_{secure'_3}$$

Simply choose $S = R$ and $S = F - \{x\}$ in $reads_{secure_3}$

(b)

$$reads_{secure'_3} \;\Rightarrow\; reads_{secure''_3}$$

Can be seen since given $\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1$ and $S \supseteq R$ such that $\overleftarrow{\sigma_1}\,\Xi_S\,\overleftarrow{\sigma_2}$ and $\overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1$ by $reads_{secure'_3}$ we have $\sigma_2$ s.t.

$$\forall x \in S - R \cdot \sigma_1 \Xi_{R \cup \{x\}} \sigma_2 \wedge \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2$$

so $\sigma_1 \Xi_S \sigma_2$ as required.

(c)

$$reads_{secure''_3} \;\Rightarrow\; reads_{secure_3}$$

Let $\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \sigma_1$ such that $\overleftarrow{\sigma_1}\,\Xi_R\,\overleftarrow{\sigma_2}$ and $\overleftarrow{\sigma_1} \xrightarrow{\circ} \sigma_1$ be given.

Define

$$S_0 \triangleq \{x \in F \mid \overleftarrow{\sigma_1}\,\Xi_{\{x\}}\,\overleftarrow{\sigma_2}\}$$

Now by $reads_{secure''_3}$ we have

$$\exists \sigma_2 \cdot \sigma_1 \Xi_{S_0} \sigma_2 \overleftarrow{\sigma_2} \xrightarrow{\circ} \sigma_2$$

Now let $S \supseteq R$ be given, if $S \supset S_0$ then $\overleftarrow{\sigma_1}\,\not\Xi_S\,\overleftarrow{\sigma_2}$ and we are done, else if $S \subseteq S_0$ then $\sigma_1 \Xi_S \sigma_2$ as required.

### A.4.4 Non-interference with secure semantics (Section 7.2.6)

To show

$$M_2;M_1 = M_1;M_2$$

using

$$reads(R, M_i) \quad \triangleq \quad \forall S \supseteq R \cdot \Xi_S;M_i = M_i;\Xi_S$$

but without the assumption that $R_i \supseteq W_i$.

Firstly

$$
\begin{array}{llll}
(\overleftarrow{\sigma_2}, \sigma_1) \in M_1;M_2 & \Rightarrow & \exists \overleftarrow{\sigma_1} \cdot \overleftarrow{\sigma_2}\ M_1\ \overleftarrow{\sigma_1}\ M_2\ \sigma_1 & \text{def'n} \\
 & \Rightarrow & \exists \overleftarrow{\sigma_1} \cdot \overleftarrow{\sigma_2}\ \Xi_{F-W_1}\ \overleftarrow{\sigma_1}\ M_2\ \sigma_1 & \mathit{writes}(W_1, M_1) \\
 & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \overleftarrow{\sigma_2}\ M_2\ \sigma_2\ \Xi_{F-W_1}\ \sigma_1 & \\
 & & & \mathit{reads}(R_2, M_2), R_2 \subseteq F - W_1 \\
 & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2 \cdot \sigma_2\ \Xi_{F-W_2}\ \overleftarrow{\sigma_2}\ M_1\ \overleftarrow{\sigma_1} & \text{and intro} \\
 & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2, \sigma_1' \cdot \sigma_2\ M_1\ \sigma_1'\ \Xi_{F-W_2}\ \overleftarrow{\sigma_1} & \\
 & & & \mathit{reads}(R_1, M_1), R_1 \subseteq F - W_2 \\
 & \Rightarrow & \exists \overleftarrow{\sigma_1}, \sigma_2, \sigma_1' \cdot \overleftarrow{\sigma_2}\ M_2\ \sigma_2\ M_1\ \sigma_1' & \text{and elim/intro} \\
 & \Rightarrow & (\overleftarrow{\sigma_2}, \sigma_1') \in M_2;M_1 & \text{def'n}
\end{array}
$$

But $\sigma_1' = \sigma_1$ since

$$\sigma_1'\ \Xi_{F-W_2}\ \overleftarrow{\sigma_1}\ \Xi_{F-W_2}\ \sigma_1 \text{ and } \sigma_1'\ \Xi_{F-W_1}\ \sigma_2\ \Xi_{F-W_1}\ \sigma_1$$

so

$$\sigma_1'\ \Xi_{F-(W_1 \cap W_2)}\ \sigma_1$$

and $W_1 \cap W_2 = \{\,\}$.

The proof for the termination sets is the same as that given earlier (Section A.3.4).

Note that the proof with the Chapter 6 version of *reads* requires the assumption that $R_i \supseteq W_i$ (or at least $M \subseteq \Xi_{F-(R_1 \cup R_2)}$), whereas with the Chapter 7 version we do not need this (since we get $F - W_1 \cup F - W_2 = F$ from $W_1 \cap W_2 = \{\,\}$).

### A.4.5 Well-definedness of sequence semantics (Section 7.3.3)

For each $f: \mathcal{D}_c \to \mathcal{D}_a$ we require

1) $f$ is a total surjection,

2) $f$ is consistent. i.e. for all $Op$, $f[\![Op]\!]_{\mathcal{D}_c} = [\![Op]\!]_{\mathcal{D}_a}$,

3) $f$ preserves refinement, i.e.

$$\frac{Op_1 \sqsubseteq_{\mathcal{D}_c} Op_1}{Op_1 \sqsubseteq_{\mathcal{D}_a} Op_1}$$

The case $retr: \mathcal{D}_1 \to \mathcal{D}_0$ has been given in detail in Chapter 6. The other cases follow along similar lines and raise no further points of interest. For the sake of brevity, only the required constructions and definitions are given here.

Totality is straightforward in each case. For surjectivity: given $N: \mathcal{P}\Sigma^a$, take $\mathcal{S}_N = \{N\}$; given $\mathcal{S} = \{(T_i, M_i) \mid i \in S\}$, take $\mathcal{S}_N = \{\mathcal{N}(T_i, M_i) \mid (T_i, M_i) \in \mathcal{S}\}$; and given $(T, M): \mathcal{D}_0$, take $\mathcal{N}(T, M)$.

For preservation of refinement we define refinement in $\mathcal{P}\Sigma^a$ as follows

$$N_1 \sqsubseteq N_2 \quad \triangleq \quad \mathcal{T}(N_1) \subseteq \mathcal{T}(N_2) \wedge \mathcal{M}(N_1) \supseteq \mathcal{T}(N_1) \lhd \mathcal{M}(N_2)$$

the results then follows easily.

For consistency, we require a definition of the sequence based semantics for each explicit operation constructor. These can be given as follows:

$$\begin{array}{lcl}
[\![\textsf{if } B \textsf{ then } Op_1 \textsf{ else } Op_2]\!]_N & \triangleq & \{I(B, N_1, N_2) \mid N_i \in [\![Op_i]\!]_N\} \\
[\![Op_1 ; Op_2]\!]_N & \triangleq & \{S(N_1, N_2) \mid N_i \in [\![Op_i]\!]_N\} \\
[\![x := e]\!]_N & \triangleq & \{A(x, e)\} \\
[\![\textsf{while } B \textsf{ do } Op_1 \textsf{ end}]\!]_N & \triangleq & \{W(B, N) \mid N \in [\![Op]\!]_N\}
\end{array}$$

where

$$\begin{array}{lcl}
I(B,N_1,N_2) & \triangleq & \overline{B} \lhd N_1 \bigcup \overline{\neg B} \lhd N_2 \\
S(N_1, N_2) & \triangleq & N_1 ; N_2 \quad \text{(See Section A.4.7 for definition.)} \\
A(x, e) & \triangleq & \{[\sigma, \sigma[e/x]] \mid \sigma \in \Sigma\} \\
W(B, N) & \triangleq & \left\{[\sigma] \;\middle|\; \sigma \in \Sigma - \overline{B}\right\} \bigcup \\
& & \left\{ \sigma_1^f \frown \ldots \frown \sigma_k^f \frown \sigma_{k+1}^f \;\middle|\; \begin{array}{l} k \in \mathbb{N} \wedge \\ \forall i \in \{1, \ldots, k\} \cdot \sigma_i^f \in \overline{B} \lhd N^f \rhd \overline{B} \wedge \\ \sigma_{k+1}^a \in (\overline{B} \lhd N^f \rhd \overline{\neg B}) \bigcup (\overline{B} \lhd N^i) \end{array} \right\}
\end{array}$$

### A.4.6 Feasibility in sequence semantics (Section 7.3.3)

Let $N : \mathcal{P}\Sigma^a$, $T : \mathcal{P}\Sigma$ and $M : \mathcal{P}(\Sigma \times \Sigma)$ such that $T = \mathcal{T}(N)$ and $M = \mathcal{M}(N)$ be given. Then we have

$$\textsf{dom}\, N = \Sigma \quad \Leftrightarrow \quad T \subseteq \textsf{dom}\, M$$

**Proof**

$$\begin{array}{rcl}
T \subseteq \mathsf{dom}\, M & \Leftrightarrow & \forall \sigma{:}\, \Sigma \cdot (\sigma \in T) \;\Rightarrow\; (\sigma \in \mathsf{dom}\, M) \\
& \Leftrightarrow & \quad \forall \sigma{:}\, \Sigma \cdot (\sigma \in \Sigma - \{\sigma^i(1) \mid \sigma^i \in N\}) \\
& & \qquad\qquad \Rightarrow\; (\sigma \in \mathsf{dom}\, \{(\sigma^f(1), \sigma^f(\mathsf{len}\, \sigma^f)) \mid \sigma^f \in N\}) \\
& \Leftrightarrow & \quad \forall \sigma{:}\, \Sigma \cdot (\sigma \notin \{\sigma^i(1) \mid \sigma^i \in N\}) \\
& & \qquad\qquad \Rightarrow\; (\sigma \in \mathsf{dom}\, \{(\sigma^f(1), \sigma^f(\mathsf{len}\, \sigma^f)) \mid \sigma^f \in N\}) \\
& \Leftrightarrow & \forall \sigma{:}\, \Sigma \cdot (\forall \sigma^i \in N \cdot \sigma^i(1) \neq \sigma \;\Rightarrow\; \exists \sigma^f \in N \cdot \sigma^f(1) = \sigma) \\
& \Leftrightarrow & \forall \sigma{:}\, \Sigma \cdot (\forall \sigma^i \in N \cdot \sigma^i(1) \neq \sigma) \;\Rightarrow\; (\exists \sigma^f \in N \cdot \sigma^f(1) = \sigma) \\
& \Leftrightarrow & \forall \sigma{:}\, \Sigma \cdot (\exists \sigma^i \in N \cdot \sigma^i(1) = \sigma) \vee (\exists \sigma^f \in N \cdot \sigma^f(1) = \sigma) \\
& \Leftrightarrow & \forall \sigma{:}\, \Sigma \cdot \sigma \in \mathsf{dom}\, N \\
& \Leftrightarrow & \mathsf{dom}\, N = \Sigma
\end{array}$$

### A.4.7 A theory of finite and infinite sequences (Section 7.3.1)

This section presents some results about sequences that mimic results about relations. Throughout this section let $N, N_i{:}\, \mathcal{P}\Sigma^a$.

**Definition** fuse

Define a modified form of concatenation for non-empty sequences, called *fuse*, which concatenates by merging the last element of its first argument with the first element of its second argument[1]. It is undefined if these two elements are different.

$$\begin{array}{l}
_ \curvearrowright _ : \Sigma^f \times \Sigma^a \to \Sigma^a \\
\sigma_1^f \curvearrowright \sigma_2^a = \sigma_1^f \frown \mathsf{tl}\, \sigma_1^a \\
\mathsf{pre}\ \mathsf{lt}\, \sigma_1^f = \mathsf{hd}\, \sigma_2^a
\end{array}$$

Note: all sequences are of length at least 1 so head and last are defined.

So, for example, $[1,2] \curvearrowright [2,3] = [1,2,3]$ but $[1,2] \curvearrowright [3,4]$ is undefined.

**Definition** Sequencing Sequences

Define

$$\begin{array}{rcl}
N_1 ; N_2 & \triangleq & \{\sigma^i \in N_1^i\} \cup \\
& & \{\sigma_1^f \curvearrowright \sigma_2^a \mid \sigma_1^f \in N_1^f \wedge \sigma_2^a \in N_2 \wedge \mathsf{lt}\, \sigma_1^f = \mathsf{hd}\, \sigma_2^a\}
\end{array}$$

Considering the sequences as the states visited by an execution, this means that if the first execution can be non-terminating, then this is also a possible behaviour of the composition. On the other hand, if the first sequence terminates and ends with the same element with which the second starts, then fuse the two sequences together[2].

---

[1]Recall, last, $\mathsf{lt}$, returns the last element of a finite sequence, whereas tail, $\mathsf{tl}$, returns the sequence without its first element.

[2]An alternative approach would be to model a sequence of actions as a sequence of pairs where the

**Theorem** Sequencing is transitive

$$(N_1;N_2);N_3 = N_1;(N_2;N_3)$$

**Theorem** Identity for sequencing

Let $[S] \quad \triangleq \quad \{[s] \mid s \in S\}$ then

$$[\Sigma];N = N \quad \text{and} \quad N;[\Sigma] = N$$

**Definition** Domain and range

Define

$$\mathsf{dom}\, N = \{\sigma : \Sigma \mid \exists \sigma^a \in N \cdot \sigma = \sigma^a(1)\}$$

$$\mathsf{rng}\, N = \{\sigma : \Sigma \mid \exists \sigma^f \in N^f \cdot \sigma = \sigma^f(\mathsf{len}\, \sigma^f)\}$$

Note that in $\mathsf{rng}$ we take only the finite sequences.

**Theorem** Domain is distributive

$$\mathsf{dom}\,(N_1 \cup N_2) = \mathsf{dom}\, N_1 \cup \mathsf{dom}\, N_2$$

**Definition** Domain and range restriction

Let $T : \mathcal{P}\Sigma$. Define

$$T \lhd N = \{\sigma^a \in N \mid \sigma^a(1) \in T\}$$

$$N \rhd T = \{\sigma^f \in N^f \mid \sigma^f(\mathsf{len}\, \sigma^f) \in T\} \cup \{\sigma^i \in N^i\}$$

**Defintion** Restriction

Let $I : \mathcal{P}\Sigma$. Define

$$N \dagger I = \{\sigma^a \in N \mid \forall i \in \mathsf{inds}\, \sigma^a \cdot \sigma^a(i) \in I\}$$

**Theorem** Containment of restrictions

$$T \lhd N \supseteq T \dagger N \subseteq N \rhd T$$

**Theorem** Restriction is sequencing

$$T \lhd N = [T];N \quad \text{and} \quad N \rhd T = N;[T]$$

---

second component of each pair was the first component of the next. Then sequencing would be achieved by ordinary concatenation. However, we would still need to ensure that the sequence to concatenate were "touching".

**Theorem** Restriction and intersection

$$\mathsf{dom}\,(T \lhd N) = T \cap (\mathsf{dom}\,N) \quad \text{and} \quad \mathsf{rng}\,(N \rhd T) = (\mathsf{rng}\,N) \cap T$$

**Theorem** Domain restriction distributes through sequencing

$$T \lhd (N_1;N_2) = (T \lhd N_1);N_2 \quad \text{and} \quad (N_1;N_2) \rhd T = N_1;(N_2 \rhd T)$$

### A.4.8 Manipulating the overall frame (Section 7.4)

**lemma**

For $i = 1, 2$, given

$$Op_i = mk\text{-}OpDef(F_i, I_i, R_i, W_i, P_i, Q_i)$$

such that $\mathsf{dom}\,F_2 \cap \mathsf{dom}\,F_1 = \{\,\}$, let

$$Op = mk\text{-}OpDef(F_1 \cup F_2, I_1 \wedge I_2, R_1 \cup R_2, W_1 \cup W_2, P_1 \wedge P_2, Q_1 \wedge Q_2)$$

then we have

(1) $drwr_{Op_1}(T_i, M_i) \wedge drwr_{Op_2}(T_i, M_i) \;\Rightarrow\; drwr_{Op}(T_1{\times}T_2, M_1{\times}M_2)$

Furthermore, when $I_2 = P_2 = \mathsf{true}$, we also have

(2) $drwr_{Op}(T, M) \;\Rightarrow\; drwr_{Op_1}(T\big|_{F_1}, M\big|_{F_1})$

**Notation**

Now that the overall frame is no longer fixed, when taking characteristic sets it will be helpful to indicate the frame with respect to which they are taken. Define

$$\overline{P}^F \;\triangleq\; \{\sigma{:}\,F \mid P(\sigma)\}$$

**proof**

We show the following eight results

$$\bigwedge_{i=1,2} dom(\overline{P_i}^{F_i}, T_i) \;\Rightarrow\; dom(\overline{P_1{\wedge}P_2}^{F_1\cup F_2}, T_1{\times}T_2)$$

$$\bigwedge_{i=1,2} res(\overline{P_i}^{F_i}, \overline{Q_i}, M_i) \;\Rightarrow\; res(\overline{P_1{\wedge}P_2}^{F_1\cup F_2}, \overline{Q_1{\wedge}Q_2}^{F_1\cup F_2}, M_1{\times}M_2)$$

$$\bigwedge_{i=1,2} writes(W_i, M_i) \;\Rightarrow\; writes(W_1 \cup W_2, M_1{\times}M_2)$$

$$\bigwedge_{i=1,2} reads(R_i, M_i) \;\Rightarrow\; reads(R_1 \cup R_2, M_1{\times}M_2)$$

$$dom(\overline{P_1 \wedge P_2}^{F_1 \cup F_2}, T) \;\Rightarrow\; \bigwedge_{i=1,2} dom(\overline{P_i}^{F_i}, T\Big|_{F_i})$$
$$res(\overline{P_1 \wedge P_2}^{F_1 \cup F_2}, \overline{Q_1 \wedge Q_2}^{F_1 \cup F_2}, M) \;\Rightarrow \bigwedge_{i=1,2} res(\overline{P_1}^{F_i}, \overline{Q_1}^{F_i}, M\Big|_{F_i}) \text{ if } P_2 = \mathsf{true}$$
$$writes(W_1 \cup W_2, M) \;\Rightarrow\; \bigwedge_{i=1,2} writes(W_i, M\Big|_{F_i})$$
$$reads(R_1 \cup R_2, M) \;\Rightarrow\; \bigwedge_{i=1,2} reads(R_i, M\Big|_{F_i})$$

then the result follows by substituting $P \wedge I$ for $P$ and $Q \wedge I$ for $Q$.

**dom**

$$\bigwedge_{i=1,2} \overline{P_i}^{F_i} \subseteq T_i \;\Rightarrow\; \overline{P_1 \wedge P_2}^{F_1 \cup F_2} \subseteq T_1 \times T_2$$

and

$$\overline{P_1 \wedge P_2}^{F_1 \cup F_2} \subseteq T \;\Rightarrow\; \bigwedge_{i=1,2} \overline{P_i}^{F_1} \subseteq T\Big|_{F_1}$$

follow by monotonicity of Cartesian product and projection.

**res**

$$\bigwedge_{i=1,2} \overline{P_i}^{F_i} \lhd M_1 \subseteq \overline{Q_i}^{F_i} \;\Rightarrow\; \overline{P_1 \wedge P_2}^{F_1 \cup F_2} \lhd M_1 \times M_2 \subseteq \overline{Q_1 \wedge Q_2}^{F_1 \cup F_2}$$

since

$$\begin{aligned}
\overline{P_1 \wedge P_2}^{F_1 \cup F_2} \lhd M_1 \times M_2 &= \bigwedge_{i=1,2} \overline{P_i}^{F_i} \lhd M_1 \\
&\subseteq \bigwedge_{i=1,2} \overline{Q_i}^{F_i} \\
&= \overline{Q_1 \wedge Q_2}^{F_1 \cup F_2}
\end{aligned}$$

Provided $P_2 = \mathsf{true}$ we have

$$\overline{P_1 \wedge P_2}^{F_1 \cup F_2} \lhd M \subseteq \overline{Q_1 \wedge Q_2}^{F_1 \cup F_2} \;\Rightarrow\; \overline{P_1}^{F_1} \lhd M\Big|_{F_1} \subseteq \overline{Q_1}^{F_1}$$

since

$$\begin{aligned}
\overline{P_1}^{F_1} \lhd M\Big|_{F_1} &= (\overline{P_1}^{F_1} \lhd M)\Big|_{F_1} && \text{provided } P_1 = \mathsf{true} \\
&\subseteq \overline{Q_1}^{F_1} \times \overline{Q_2}^{F_2} \\
&= (\overline{Q_1 \wedge Q_2}^{F_1 \cup F_2})\Big|_{F_1}
\end{aligned}$$

**writes**

$$
\begin{aligned}
\bigwedge_{i=1,2} M_i \subseteq \Xi_{F_i - W_i} &\Rightarrow M_1{\times}M_2 \subseteq \Xi_{F_1 - W_1} \times \Xi_{F_2 - W_2} \\
&\Rightarrow M_1{\times}M_2 \subseteq \Xi_{(F_1 - W_1)\cup(F_2 - W_2)} \text{ since } W_i \subseteq F_i \\
&\Rightarrow M_1{\times}M_2 \subseteq \Xi_{(F_1 \cup F_2)-(W_1 \cup W_2)}
\end{aligned}
$$

$$
\begin{aligned}
M \subseteq \Xi_{(F_1 \cup F_2)-(W_1 \cup W_2)} &\Rightarrow M \subseteq \Xi_{(F_1 - W_1)\cup(F_2 - W_2)} \text{ since } W_i \subseteq F_i \\
&\Rightarrow M \subseteq \Xi_{F_1 - W_1} \times \Xi_{F_2 - W_2} \\
&\Rightarrow \bigwedge_{i=1,2} M\big|_{F_i} \subseteq \Xi_{F_i - W_i}
\end{aligned}
$$

**reads**

(a) r.t.p.

$$\bigwedge_{i=1,2} \Xi_{R_i};M_i \subseteq M_i;\Xi_{R_i} \Rightarrow \Xi_{R_1 \cup R_2};M_1{\times}M_2 \subseteq M_1{\times}M_2;\Xi_{R_1 \cup R_2}$$

Now this follows as in previous *reads* proofs since

$$
\begin{aligned}
\overleftarrow{\sigma_2}\; \Xi_{R_1 \cup R_2}\; \overleftarrow{\sigma_1}\; M_1{\times}M_2\; \sigma_1 &\Rightarrow \overleftarrow{\sigma_2}\big|_{F_i} \Xi_{R_i}\; \overleftarrow{\sigma_1}\big|_{F_i} M_1{\times}M_2\; \overleftarrow{\sigma_1}\big|_{F_i} \\
&\Rightarrow \overleftarrow{\sigma_2}\big|_{F_i} M_i\; \sigma_2\big|_{F_i} \Xi_{R_i}\; \overleftarrow{\sigma_1}\big|_{F_i} \quad \text{for some } \sigma_2\big|_{F_i},\ i = 1, 2 \\
&\Rightarrow \overleftarrow{\sigma_2}\; M_1{\times}M_2\; \sigma_2\; \Xi_{R_1 \cup R_2}\; \overleftarrow{\sigma_1}
\end{aligned}
$$

(b) r.t.p.

$$\Xi_{R_1 \cup R_2};M \subseteq M;\Xi_{R_1 \cup R_2} \Rightarrow \bigwedge_{i=1,2} \Xi_{R_i};M\big|_{F_i} \subseteq M\big|_{F_i};\Xi_{R_i}$$

w.l.o.g. take the case $i = 1$,

$$
\begin{aligned}
&\overleftarrow{\sigma_2}\big|_{F_1} \Xi_{R_1}\; \overleftarrow{\sigma_1}\big|_{F_1} M\big|_{F_1} \sigma_1\big|_{F_1} \\
&\Rightarrow (\overleftarrow{\sigma_2}\big|_{F_1}, \overleftarrow{\sigma_1}\big|_{F_2})\; \Xi_{R_1 \cup R_2}\; (\overleftarrow{\sigma_1}\big|_{F_1}, \overleftarrow{\sigma_1}\big|_{F_2})\; M\; (\sigma_1\big|_{F_1}, \sigma_1\big|_{F_2}) \text{for some } \overleftarrow{\sigma_1}\big|_{F_2}, \sigma_1\big|_{F_2} \\
&\Rightarrow (\overleftarrow{\sigma_2}\big|_{F_1}, \overleftarrow{\sigma_1}\big|_{F_2})\; M\; (\sigma_2\big|_{F_1}, \sigma_2\big|_{F_2})\; \Xi_{R_1 \cup R_2}\; (\sigma_1\big|_{F_1}, \sigma_1\big|_{F_2}) \text{for some } \sigma_2 \\
&\Rightarrow \overleftarrow{\sigma_2}\big|_{F_1} M\big|_{F_1} \sigma_2\big|_{F_1} \Xi_{R_1}\; \sigma_1\big|_{F_1} \text{ by following lemma}
\end{aligned}
$$

**lemma**

$$(R;R')\big|_{F_1} \subseteq R\big|_{F_1};R'\big|_{F_1}$$

which follows since $\overleftarrow{\sigma}\, R \sigma \Rightarrow \overleftarrow{\sigma}\big|_{F_1} R\big|_{F_1} \sigma\big|_{F_1}$