Testing Refinements by Refining Tests

John Derrick and Eerke Boiten

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
J.Derrick@ukc.ac.uk

Abstract. One of the potential benefits of formal methods is that they offer the possibility of reducing the costs of testing. A specification acts as both the benchmark against which any implementation is tested, and also as the means by which tests are generated. There has therefore been interest in developing test generation techniques from *formal* specifications, and a number of different methods have been derived for state based languages such as Z, B and VDM. However, in addition to deriving tests from a formal specification, we might wish to refine the specification further before its implementation.
The purpose of this paper is to explore the relationship between testing and refinement. As our model for test generation we use a DNF partition analysis for operations written in Z, which produces a number of disjoint test cases for each operation. In this paper we discuss how the partition analysis of an operation alters upon refinement, and we develop techniques that allow us to refine abstract tests in order to generate test cases for a refinement. To do so we use (and extend existing) methods for calculating the weakest data refinement of a specification.

Keywords: Testing; Partition Analysis; Disjunctive Normal Form; Refinement; Calculating data refinements.

1 Introduction

Testing and specifications are intrinsically interlinked. Specifications act as the benchmark against which any implementation is tested, and they also provide a means by which to generate the tests themselves. The advent and use of formal methods does not change this. Although the aim of formal methods is to move some of the effort spent on error detection to more effort spent on correct construction, even a fully verified formal development will at some stage be tested against the original specification. Indeed, the use of formal methods offers a promise of reduced overall development cost by automating part of the testing process.

There has therefore been interest in developing techniques by which test case generation and test case scheduling can be automatically (or semi-automatically) generated from formal specifications. Different paradigms have developed different ways to do this, and techniques for state based languages such as Z [17], B [1] and VDM [12] have been developed, see for example [14, 5, 8, 4, 11, 18].

There are many aspects to the provision of formal support for the testing process. In this paper we shall be concerned with the issue of test case generation from individual operations. The attraction of using an abstract formal specification as the basis to generate the tests (as opposed to an informal specification of even an implementation) is that it concisely captures the essential behaviour required: any correct implementation should pass all the tests derived from this specification, and yet the tests will be as abstract as possible, ensuring their number is kept low.

One elegant and simple method for generating and sequencing tests from state based languages has been developed by Dick and Faivre [8]. The basic technique of test generation consists of a partition analysis, which reduces the specification of each operation into its Disjunctive Normal Form (DNF). The approach was based on VDM, but has been applied to Z in [11, 15] and B in [20], and benefits from tool support, which is described in [8] and [20]. [11] describes an industrial application of the method to an aircraft control system.

However, in addition to deriving tests from a formal specification, we might wish to develop or refine the specification further before its implementation. Indeed we can view any implementation as a refinement of the original specification. The conditions under which a development is a correct refinement are encapsulated into two refinement rules: downward and upward simulations [22]. To verify a refinement the simulations use a retrieve relation which relates the concrete to abstract states.

The purpose of this paper is to explore the relationship between testing and refinement. In particular, we aim to develop techniques whereby we can reuse abstract tests to develop tests for a concrete specification or implementation. As our model for test generation we use the DNF partition analysis for operations written in Z as discussed in [11], although it should be noted that the methods are applicable to other testing scenarios and state based languages such as B and VDM.

Dick and Faivre did not consider further refinements of the abstract specification, however, they posed the open question: *does refining a specification create a super-set of the partitions of the previous level?* We will answer this question in the negative. We will then go onto answer the question: *how do we generate tests for a refinement based on the tests derived from the abstract specification?* We do so by developing a means to calculate concrete tests based upon methods that generate the weakest (i.e. most general) refinement of an abstract operation. We do this first for refinements which are downward simulations, and we discuss the properties of the constructed tests, and in particular whether they capture all the requirements and whether they are disjoint. We next develop similar results for upward simulations, however, here we first have to derive techniques to calculate the weakest upward simulation of an operation. In each case the results simplify if the retrieve relation used in the refinement is a surjective function from concrete to abstract state spaces.

The structure of the paper is as follows. Section 2 introduces the method of DNF partition analysis, and Section 3 provides some background material on refinement in Z. Sections 4 and 5 form the heart of the paper where we develop the theory of testing refinements by refining tests, and discuss relevant properties. Section 4 looks at downward simulations and Section 5 considers upward simulations. We conclude in Section 6.

2 Testing

Testing is an indespensible part of the software construction and maintenance process, irrespective of whether or not the development of a system has involved the use of formal methods and verification. Therefore there has been considerable interest in the use of formal methods to *support* the testing process as opposed to viewing formal methods as an *alternative* to the testing process [19, 5, 8, 4, 11, 18].

Different formal paradigms have associated methods for aiding this test generation process in an automatic, semi-automatic or manual fashion. For example, there has been considerable research on testing specifications in the context of process algebras [10, 6, 3, 2]. There has also been analogous work for state based languages such as Z, B and VDM. The approach we consider here is that of Dick and Faivre [8], which describes a means to automate test generation and sequencing from VDM specifications, and has also been applied to Z specifications in [11, 15]. For example, [11] describes application of this methodology to a portion of the Cabin Intercommunication Data System for the Airbus A330/340 aircraft. An alternative approach to testing is discussed in [18] which derives a testing methodology suitable for the construction of tests from OSI Managed Object specifications [21], and manual approaches to test generation have also been considered in [14, 4].

Dick and Faivre consider the complete testing activity from test generation from individual operations, through the scheduling of tests, to the verification of test results. The basic technique of test case generation consists of a partition analysis, which reduces the specification of each operation into its Disjunctive Normal Form (DNF). Each element in the DNF represents an individual test case for the operation. The partition then serves as a basis for the construction of a finite state automaton (FSA) which is then used to derive test suites (i.e. a structured sequence of test cases).

In this paper we are concerned with the use of DNFs to provide a suitable partition analysis of operations, and we aim to show how this partition alters upon refinement.

As an example of the methodology let us consider the specification of a cinema box office (adapted from [22] and [16]). The Kurbel box office allows customers to book tickets in advance by telephone. When a customer calls, if there is an available ticket then the customer's name is simply recorded. When a customer whose name has been recorded arrives at the box office, a ticket is allocated. The Kurbel is specified as follows.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{Kurbel}} \\
kpool : \mathbb{P}\, Ticket \\
bkd : \mathbb{P}\, Name \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{KInit}} \\
Kurbel' \\
\hline
bkd' = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{KBook}} \\
\Delta Kurbel \\
name? : Name \\
\hline
name? \notin bkd \\
\# bkd < \# kpool \\
bkd' = bkd \cup \{name?\} \\
kpool' = kpool \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{KArrive}} \\
\Delta Kurbel \\
name? : Name \\
t! : Ticket \\
\hline
name? \in bkd \\
bkd' = bkd \setminus \{name?\} \\
t! \in kpool \\
kpool' = kpool \setminus \{t!\} \\
\hline
\end{array}
$$

The state variable *kpool* denotes the pool of tickets and *bkd* denotes the set of names of customers who have booked a ticket. The operation *KBook* records a booking provided that there are currently less bookings than tickets. The operation *KArrive* allocates a ticket to a customer who has a booking. In order to test an implementation of the box office we generate test cases for each operation in the specification.

We do this by transforming each operation into a DNF. Each schema in this DNF then represents a single test case. Each test case will be disjoint, allowing them all to be treated separately. The transformation into test cases for *KBook* and *KArrive* is thus given by (to simplify the presentation we just consider tests for a single fixed input *name?* throughout the paper):

$KBook = KBook$
$KArrive = \bigvee_{t \in kpool} KA_t$ where

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{KA_t}} \\
\Delta Kurbel \\
name? : Name \\
t! : Ticket \\
\hline
name? \in bkd \\
bkd' = bkd \setminus \{name?\} \\
t! = t \\
kpool' = kpool \setminus \{t!\} \\
\hline
\end{array}
$$

We have used a distributed disjunction ($\bigvee$) here, which although nonstandard Z, can be defined in the obvious manner (for example, by $\exists\, t : kpool \bullet KA_t$). Similarly, the equality sign between schemas should be viewed as schema equivalence. We retain $\bigvee$ and $=$ for the sake of clarity.

From this we see that *KBook* is already in DNF, and thus represents a single atomic test case in itself. However, *KArrive* has a number of test cases, each one representing a different possible choice of allocated ticket. This structuring of test cases as DNFs has two important properties: coverage and disjointness; that is, *KArrive* equals the disjunction of its test cases (coverage) and these tests are disjoint. In general we say that a collection of tests $\{AOp_i\}_i$ covers an operation AOp acting on state space *Astate* if

$$AOp = \bigvee_i AOp_i$$

and that the tests are disjoint, if, for all $i \neq j$

$$\neg \exists \, Astate; \; Astate' \bullet AOp_i \wedge AOp_j$$

It is easy to see that $\{KA_t\}_{t \in kpool}$ form a disjoint covering of *KArrive*.

Note that there are many possible decompositions of an operation into DNF, and not every decomposition will produce test cases considering single elements $t \in kpool$. For example, if *kpool* was infinite some of the test cases would contain infinite partitions of *kpool* representing the various test cases we are interested in. It is by this means that a finite state machine can be obtained from a specification with infinite state. See [8] for a discussion of this point.

3 Refinement

In addition to deriving tests from a formal specification, we might wish to *refine* the specification further before its implementation. Such a refinement might typically weaken the precondition of an operation, remove some non-determinism or even alter the state space of the specification. The conditions under which a development is a correct refinement are encapsulated into two rules: downward and upward simulations [22]. These refinement rules are known to be sound and jointly complete, that is any upward or downward simulation is a valid refinement, and any refinement can be proved correct by application of appropriate upward and downward simulations [9, 23]. (Downward and upward simulations are sometimes also known as forward and backward simulations respectively.)

The downward simulation rules are more straightforward, and form the usual presentation of refinement (e.g. as in [17]), however, upward simulations are occasionally necessary, for example when the resolution of non-determinism has been postponed [22]. Let us consider an abstract specification with state space *Astate* and initialisation schema *Ainit* being refined by a concrete specification with state space *Cstate* and initialisation schema *Cinit*.

Definition 1. *Downward simulation*
The concrete specification is a downward simulation of the abstract if there is a retrieve relation Ret such that every abstract operation AOp is recast into a concrete operation COp and the following hold.

DS.1 $\forall \, Astate; \; Cstate \bullet \mathrm{pre}\, AOp \wedge Ret \implies \mathrm{pre}\, COp$
DS.2 $\forall \, Astate; \; Cstate; \; Cstate' \bullet Ret \wedge \mathrm{pre}\, AOp \wedge COp \implies \exists \, Astate' \bullet Ret' \wedge AOp$

DS.3 $\forall\, Cstate' \bullet Cinit \implies \exists\, Astate' \bullet Ainit \land Ret$

Definition 2. *Upward simulation*
The concrete specification is an upward simulation of the abstract if there is a retrieve relation Ret such that every abstract operation AOp is recast into a concrete operation COp and the following hold.

US.1 $\forall\, Cstate \bullet (\forall\, Astate \bullet Ret \implies \mathrm{pre}\, AOp) \implies \mathrm{pre}\, COp$
US.2 $\forall\, Astate';\ Cstate;\ Cstate' \bullet (\forall\, Astate \bullet Ret \implies \mathrm{pre}\, AOp) \implies (COp \land Ret' \implies \exists\, Astate \bullet Ret \land AOp)$
US.3 $\forall\, Astate';\ Cstate' \bullet Cinit \land Ret' \implies Ainit$

As an example, consider the specification of the Marlowe box office. Like the Kurbel, the Marlowe box office allows customers to book tickets in advance by telephone. However, the procedure is different from that used at the Kurbel. When a customer calls, if there is an available ticket then one is allocated and put to one side for the caller. When the customer arrives, they are presented with this ticket.

$$\begin{array}{|l}
\multicolumn{1}{l}{Marlowe} \\ \hline
mpool : \mathbb{P}\, Ticket \\
tkt : Name \rightarrowtail Ticket \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{MInit} \\ \hline
Marlowe' \\ \hline
tkt = \varnothing \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{MBook} \\ \hline
\Delta Marlowe \\
name? : Name \\ \hline
name? \notin \mathrm{dom}\, tkt \\
mpool \neq \varnothing \\
\exists\, t : mpool \bullet \\
\quad mpool' = mpool \setminus \{t\} \\
\quad tkt' = tkt \cup \{name? \mapsto t\} \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{MArrive} \\ \hline
\Delta Marlowe \\
name? : Name \\
t! : Ticket \\ \hline
name? \in \mathrm{dom}\, tkt \\
t! = tkt(name?) \\
tkt' = \{name?\} \lhd tkt \\
mpool' = mpool \\ \hline
\end{array}$$

The contrast between the Marlowe and the Kurbel box offices is the point of allocation of tickets (at booking time *vs* at collection time). However, at this level of abstraction the customer cannot tell that the Kurbel is behaving differently to the Marlowe, and this can be demonstrated by showing (see [22]) that the Marlowe is a downward simulation of the Kurbel where the retrieve relation is given by

$$\begin{array}{|l}
\multicolumn{1}{l}{Ret} \\ \hline
Kurbel \\
Marlowe \\ \hline
bkd = \mathrm{dom}\, tkt \\
kpool = mpool \cup \mathrm{ran}\, tkt \\
mpool \cap \mathrm{ran}\, tkt = \varnothing \\ \hline
\end{array}$$

In fact, the Kurbel specification is also a refinement of the Marlowe, but this must be shown using an upward simulation (i.e. it is *not* a downwards simulation), where we use the same retrieve relation as before. Therefore the Marlowe and Kurbel have identical observational behaviour, and so the tests for one specification should be able to be applied to the other. In order to do this and to be able to reuse abstract tests to test a refinement we have to be able to translate the state spaces of each test case, and we will use the retrieve relation to do this. This will involve us *calculating* refinements, a process that we now describe.

3.1 Calculating Downward Simulations

Given an abstract specification, a concrete state space and a retrieve relation between the concrete and abstract state spaces, it is possible to calculate the weakest (most general) description of the concrete operations [13, 22]. Let $Astate$ and $Cstate$ be the abstract and concrete state spaces, Ret the retrieve relation and AOp an abstract operation. We calculate* the weakest refinement COp of AOp by

$$\begin{aligned} COp \mathrel{\widehat=} & (\exists\, Astate \bullet \operatorname{pre} AOp \land Ret) \land \\ & \qquad (\forall\, Astate \bullet \operatorname{pre} AOp \land Ret \Rightarrow \exists\, Astate' \bullet AOp \land Ret') \end{aligned}$$

In general, if it is not known whether Ret defines a refinement, it is necessary to check the applicability. This is summarised in the following theorem (for a proof see [13]) which shows that COp is the weakest refinement of AOp, provided that one exists.

Theorem 1. *Let us denote a downward simulation by $\sqsubseteq_{DS}$. Suppose that AOp specifies an operation over the abstract state space $Astate$. Let $Cstate$ be a concrete state space, and Ret a retrieve relation between concrete and abstract. Let COp be defined as above. Then for every operation X*

$$AOp \sqsubseteq_{DS} X \ \textit{ iff } \ \textit{pre}\, AOp \land Ret \Rightarrow \textit{pre}\, COp \ \textit{ and } \ COp \sqsubseteq_{DS} X$$

We are interested in cases when it is known that Ret defines a refinement since *we are generating tests for an existing development*, therefore we know that applicability ($\operatorname{pre} AOp \land Ret \Rightarrow \operatorname{pre} COp$) holds. In these circumstances COp describes our most general concrete refinement of the operation AOp.

The calculation can be simplified considerably ([13, 22]) when the retrieve relation defines a surjective (partial) function from $Cstate$ to $Astate$, and we find that the following suffices for COp.

$$COp \mathrel{\widehat=} \exists\, Astate;\ Astate' \bullet Ret \land AOp \land Ret'$$

* We use calculate in the sense that COp is described by a formula in terms of known components. One might also say that COp is specified instead of calculated, and that the specification of COp is the starting point for its calculation through a series of simplification steps.

For example, the retrieve relation from Marlowe to Kurbel could in fact be used to calculate the book and arrive operations in Marlowe. The retrieve relation is functional since both *kpool* and *bkd* are uniquely determined by *Ret*, however, *Ret* is not surjective (states where $\#bkd > \#kpool$ are not in the range of *Ret*). We can in fact make it surjective without altering the specification by adding the state invariant $\#bkd \leq \#kpool$ to Kurbel, the simplified method of calculation can then be used.

In fact it can be shown [7] that the complex formula given in Theorem 1 can *always* be replaced by the simplified version $\exists Astate; \ Astate' \bullet Ret \wedge AOp \wedge Ret'$. We will therefore use this simplified version subsequently.

The method described in [13, 22] calculates the weakest downward simulation. We shall derive similar results for upward simulations in Section 5.1 below.

3.2 Generating Tests

The technique we develop for generating tests for a refinement is very simple. Given an abstract specification with operation AOp and a covering disjoint set of tests $\{AOp_i\}_i$; a concrete specification with operation COp which refines AOp, and a retrieve relation Ret, we generate a set of tests $\{COp_i\}_i$ where each test COp_i is the weakest refinement calculated from Ret and AOp_i. The remainder of the paper discusses the two cases of downward and upward simulations separately, and each case is subdivided according as to whether Ret is a surjective function or not. In each case we explore the two questions:

- do the tests $\{COp_i\}_i$ cover COp;
- are the tests $\{COp_i\}_i$ disjoint.

4 Refining Tests 1: Downward Simulations

Downward simulations are perhaps the most common form of state based refinement: we saw an example above where the Marlowe box office was a downward simulation of the Kurbel box office. How do the test cases of the operations in the two specifications compare, and in particular *does refining a specification create a super-set of the partitions of the previous level?* [8]. To answer this question let us derive the test cases for the Marlowe operations:

$MArrive = MArrive$
$MBook = \bigvee_{t \in mpool} MB_t$ where

$$\begin{array}{|l}
\multicolumn{1}{l}{MB_t} \\
\hline
\Delta Marlowe \\
name? : Name \\
\hline
name? \notin \mathrm{dom}\, tkt \\
mpool \neq \varnothing \\
mpool' = mpool \setminus \{t\} \\
tkt' = tkt \cup \{name? \mapsto t\} \\
\hline
\end{array}$$

and document the results in the following table.

	Kurbel	Marlowe
Book	$KBook$	$\bigvee_{t \in mpool} MB_t$
Arrive	$\bigvee_{t \in kpool} KA_t$	$MArrive$

From this table we see that for the book operation one test ($KBook$) becomes $\#mpool$ tests (MB_t) upon refinement, whereas for the arrive operation a collection of $\#kpool$ tests become one. This clearly answers the question of Dick and Faivre in the negative in the first instance - we do not in general create a super-set of the partition upon refinement. Let us see how calculating the tests effects coverage and disjointness in general.

4.1 Functional Surjective Retrieve Relation

We first consider the particular case when the retrieve relation used is a surjective function from concrete to abstract. Given an operation AOp with $AOp = \bigvee_i AOp_i$ being its disjoint set of tests, and a retrieve relation Ret which is a surjective function, the concrete tests are given by

$$COp_i \mathrel{\widehat=} \exists\, Astate;\ Astate' \bullet Ret \wedge AOp_i \wedge Ret'$$

These will in some way represent test cases for the original concrete operation COp, and in fact we have the following result.

Theorem 2. *Let AOp be an abstract operation with $AOp = \bigvee_i AOp_i$ being its disjoint set of tests. Let COp be a downward simulation of AOp. Let Ret be the retrieve relation. Let COp_i be the concrete tests given above. Then*

$$\bigvee_i COp_i \sqsubseteq_{DS} COp$$

and if COp is the weakest downward simulation of AOp then $COp = \bigvee_i COp_i$.

Proof

The proof is simple, and follows from:

$$\begin{aligned}\bigvee_i COp_i &= \bigvee_i (\exists\, Astate;\ Astate' \bullet Ret \wedge AOp_i \wedge Ret')\\ &= \exists\, Astate;\ Astate' \bullet \bigvee_i (Ret \wedge AOp_i \wedge Ret')\\ &= \exists\, Astate;\ Astate' \bullet Ret \wedge \bigvee_i AOp_i \wedge Ret'\\ &= \exists\, Astate;\ Astate' \bullet Ret \wedge AOp \wedge Ret'\\ &\sqsubseteq_{DS} COp\end{aligned}$$

□

The practical consequences of this is that we can use abstract tests together with the retrieve relation to calculate tests for a refinement.

Example 1. Calculating tests for a refinement.

Consider the following two specifications which describe *Staff* entering and leaving the box office. The first is specified using a set

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{SSystem}} \\ \hline s : \mathbb{P}\,\mathit{Staff} \\ \hline \#s \leq \mathit{maxentry} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{SInit}} \\ \hline \mathit{SSystem}' \\ \hline s' = \varnothing \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{SEnter}} \\ \hline \Delta \mathit{SSystem} \\ p? : \mathit{Staff} \\ \hline \#s < \mathit{maxentry} \\ p? \notin s \\ s' = s \cup \{p?\} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{SLeave}} \\ \hline \Delta \mathit{SSystem} \\ p? : \mathit{Staff} \\ \hline p? \in s \\ s' = s \setminus \{p?\} \\ \hline \end{array}$$

The second description uses a list (an injective sequence)

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{LSystem}} \\ \hline l : \mathrm{iseq}\,\mathit{Staff} \\ \hline \#l \leq \mathit{maxentry} \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{LInit}} \\ \hline \mathit{LSystem}' \\ \hline l' = \langle\,\rangle \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{LEnter}} \\ \hline \Delta \mathit{LSystem} \\ p? : \mathit{Staff} \\ \hline \#l < \mathit{maxentry} \\ p? \notin \mathrm{ran}\, l \\ l' = l \frown \langle p? \rangle \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{\textit{LLeave}} \\ \hline \Delta \mathit{LSystem} \\ p? : \mathit{Staff} \\ \hline p? \in \mathrm{ran}\, l \\ l' = l \upharpoonright (\mathit{Staff} \setminus \{p?\}) \\ \hline \end{array}$$

The second specification is a refinement of the first (see [22]), where the retrieve relation is given by

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{Ret}} \\ \hline \mathit{LSystem} \\ \mathit{SSystem} \\ \hline s = \mathrm{ran}\, l \\ \hline \end{array}$$

This is a total surjective function from concrete (list) to abstract (set). The test cases of *SEnter* are just *SEnter* itself, however, calculating the weakest refinement $\exists\, \mathit{SSystem};\ \mathit{SSystem}' \bullet \mathit{Ret} \land \mathit{SEnter} \land \mathit{Ret}'$ to give the concrete test cases produces:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{LEnter}} \\
\hline
\Delta \textit{LSystem} \\
p? : \textit{Staff} \\
\hline
\# l < \textit{maxentry} \\
p? \notin \operatorname{ran} l \\
\operatorname{ran} l' = \operatorname{ran} l \cup \{p?\} \\
\hline
\end{array}
$$

The partition of this into DNF will produce a collection of tests $\{LEnter_i\}_i$, one for each possible choice of l' satisfying $\operatorname{ran} l' = \operatorname{ran} l \cup \{p?\}$ We can see that $\bigvee_i LEnter_i \sqsubseteq_{DS} LEnter$, but since *LEnter* is not the weakest refinement of *SEnter* the calculated tests contain additional tests not included in the concrete operation.

However, in this case we can construct an exact covering by taking the individual tests to be $LEnter_i \wedge LEnter$. Indeed this is a general strategy which works whenever the concrete operation has failed to be the weakest refinement because it has resolved more non-determinism than formally necessary. $\square$

Note that from this example we can see that after calculating the concrete tests, further partition analysis might be necessary to put them into DNF.

So much for coverage, what about disjointness? For a functional surjective retrieve relation disjoint abstract tests will generate disjoint concrete tests.

Theorem 3. *Let $\{AOp_i\}_i$ be disjoint test cases, Ret a functional surjective retrieve relation and $\{COp_i\}_i$ calculated from $\{AOp_i\}_i$. Then $\{COp_i\}_i$ are disjoint.*

Proof

Suppose that $\{COp_i\}_i$ were not disjoint. Then for some i and j

$$\exists\, Cstate;\ Cstate' \bullet COp_i \wedge COp_j$$

Thus there exists states *Cstate* and *Cstate'* for which

$$\exists\, Astate;\ Astate' \bullet Ret \wedge AOp_i \wedge Ret', \quad \text{and}$$
$$\exists\, Astate;\ Astate' \bullet Ret \wedge AOp_j \wedge Ret'$$

For these states *Cstate* and *Cstate'*, there are unique states *Astate* and *Astate'* such that $Ret \wedge Ret'$. Therefore

$$\exists\, Astate;\ Astate' \bullet AOp_i \wedge AOp_j$$

and so $\{AOp_i\}_i$ are not disjoint. $\square$

Note that disjointness is not the same as inequality (two tests with false predicates are considered disjoint).

Example 2. Refined tests are disjoint.

If we consider the operation *KArrive* in the Kurbel box office and its set of tests $\{KA_t\}_{t \in kpool}$. These are disjoint and we produce a set of disjoint concrete tests

$$
\begin{array}{|l}
\hline
MA_t \\
\Delta Marlowe \\
name? : Name \\
t! : Ticket \\
\hline
name? \in \mathrm{dom}\, tkt \\
t! = t = tkt(name?) \\
tkt' = \{name?\} \ntriangleleft tkt \\
mpool' = mpool \\
\hline
\end{array}
$$

All but one of these tests are false (tkt is a function, so $tkt(name?)$ must be a unique t). Therefore the set of concrete tests $\{MA_t\}$ reduces to the single test *MArrive*. □

4.2 General Retrieve Relation

We now consider the general case. Recall that to generate tests from abstract test cases $\{AOp_i\}_i$ we can still use the simplified formula

$$COp_i \mathrel{\widehat{=}} \exists\, Astate;\ Astate' \bullet Ret \land AOp_i \land Ret'$$

Therefore in this general case the covering theorem still holds. However, disjointness in general fails as the proof needed functionality of the retrieve relation. This can be seen from the following example.

Example 3. Refined tests are not disjoint in general.

Consider the two specifications which describe staff entering and leaving the box office. Suppose that we modify the second specification so that *LEnter* is now

$$
\begin{array}{|l}
\hline
LEnter \\
\Delta LSystem \\
p? : Staff \\
\hline
\# l < maxentry \\
p? \notin \mathrm{ran}\, l \\
\mathrm{ran}\, l' = \mathrm{ran}\, l \cup \{p?\} \\
\hline
\end{array}
$$

SSystem is now a refinement of this specification with the same retrieve relation as before. However, viewed this way round the retrieve relation is not functional: each set s has many (abstract) representations as a list with $s = \operatorname{ran} l$.

The DNF for *LEnter* contains many tests (one for each permutation of l with $p?$ inserted into it); for example, two such tests would be

$$\begin{array}{|l}\hline \multicolumn{1}{l}{LEnter_1} \\ \Delta LSystem \\ p? : Staff \\ \hline \#l < maxentry \\ p? \notin \operatorname{ran} l \\ l' = l \frown \langle p? \rangle \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \multicolumn{1}{l}{LEnter_2} \\ \Delta LSystem \\ p? : Staff \\ \hline \#l < maxentry \\ p? \notin \operatorname{ran} l \\ l' = \langle p? \rangle \frown l \\ \hline \end{array}$$

Calculating the refined tests for each one of these abstract tests produces

$$\begin{array}{|l}\hline \multicolumn{1}{l}{SEnter} \\ \Delta SSystem \\ p? : Staff \\ \hline \#s < maxentry \\ p? \notin s \\ s' = s \cup \{p?\} \\ \hline \end{array}$$

in every case. So *all* the abstract tests were mapped onto the same concrete test, which are therefore not disjoint. □

5 Refining Tests 2: Upward Simulations

Some valid refinements can not be proved correct with a downwards simulation, and for these we need to use an upwards simulation. An example of this was provided above where we commented that the Kurbel box office was a refinement of the Marlowe box office, but this could only be verified using an upward simulation (see [22] for details). The previous section has discussed how to derive tests from refinements which were downward simulations, we now do the same for upward simulations, and to do so we will need to derive a method for calculating the weakest upward simulation of an abstract operation.

Let us first, however, comment upon the partitioning. We found that refining a specification doesn't create a super-set of the partitions of the previous level for refinements that were downward simulations. The same can be seen to be true for refinements that are upward simulations. From the table of tests for the Kurbel and Marlowe specifications given at the start of Section 4 we find that under an upward simulation, one abstract test (*MArrive*) becomes $\#kpool$ tests (KA_t) upon refinement, and a collection of $\#mpool$ tests (MB_t) become one. There is thus, in general, no relationship between the size of the partitioning before and after refinement for both upward and downward simulations.

We will now turn to the problem of calculating the weakest upward simulation, which will allow us to derive concrete tests from abstract ones.

5.1 Calculating Upward Simulations

The methodology given in [13, 22] calculates the most general downward simulation of an abstract operation with respect to a retrieve relation between the abstract and concrete state spaces. We do the same here for upward simulations.

In a manner similar to downward simulations, the refinement rules for upward simulations simplify considerably for a retrieve relation which is a total function from concrete to abstract. In this case it is easy to show that the correctness condition US.2

$$\begin{aligned}&\forall\, Astate';\ Cstate;\ Cstate' \bullet\\ &\quad (\forall\, Astate \bullet Ret \Longrightarrow \text{pre}\, AOp) \Longrightarrow (COp \wedge Ret' \Longrightarrow \exists\, Astate \bullet Ret \wedge AOp)\end{aligned}$$

reduces to

$$\begin{aligned}\forall\, Cstate;\ Cstate' \bullet (&\forall\, Astate \bullet Ret \Longrightarrow \text{pre}\, AOp) \wedge\\ &(COp \Longrightarrow \exists\, Astate;\ Astate' \bullet Ret \wedge AOp \wedge Ret')\end{aligned}$$

Then, if the retrieve relation is additionally surjective, the weakest refinement of AOp will again be given by

$$COp \mathrel{\widehat{=}} \exists\, Astate;\ Astate' \bullet Ret \wedge AOp \wedge Ret'$$

a formula that is identical to the downward simulation case.

Turning to the general situation (i.e. an arbitrary retrieve relation), the following will define the weakest refinement of AOp

$$\begin{aligned}&COp \mathrel{\widehat{=}}\\ &\quad (\forall\, Astate \bullet Ret \Longrightarrow \text{pre}\, AOp) \wedge \forall\, Astate' \bullet (Ret' \Longrightarrow \exists\, Astate \bullet Ret \wedge AOp)\end{aligned}$$

For an arbitrary relation R we would still have to check applicability

$$\forall\, Cstate \bullet (\forall\, Astate \bullet R \Longrightarrow \text{pre}\, AOp) \Longrightarrow \text{pre}\, COp$$

However, if we know that the retrieve relation does indeed define an upward simulation it is not necessary to check this.

Theorem 4. *Let us denote an upward simulation by $\sqsubseteq_{US}$. Suppose that AOp specifies an operation over the abstract state space Astate. Let Cstate be a concrete state space, and Ret a retrieve relation between concrete and abstract. Let COp be defined as above. Then for every operation X*

$$AOp \sqsubseteq_{US} X \ \textit{ iff } (\forall\, Astate \bullet Ret \Longrightarrow \textit{pre}\, AOp) \Longrightarrow \textit{pre}\, COp \ \textit{ and } \ COp \sqsubseteq_{US} X$$

Proof

To show that the above definition of COp does refine AOp we need to show that

$$\begin{aligned}&\forall\, Astate';\ Cstate;\ Cstate' \bullet\\ &\quad (\forall\, Astate \bullet Ret \Longrightarrow \text{pre}\, AOp) \Longrightarrow (COp \wedge Ret' \Longrightarrow \exists\, Astate \bullet Ret \wedge AOp)\end{aligned}$$

which reduces to showing that

$$\begin{array}{l} \forall\, Astate';\ Cstate;\ Cstate' \bullet \\ \quad (Ret' \implies \exists\, Astate \bullet Ret \wedge AOp) \wedge Ret' \implies (Ret' \implies \exists\, Astate \bullet Ret \wedge AOp) \end{array}$$

which can easily seen to be true.

To show that COp defines the most general refinement of AOp, let us suppose that in addition $AOp \sqsubseteq_{US} X$, we will show that $COp \sqsubseteq_{US} X$. Furthermore, let us suppose that the refinement $AOp \sqsubseteq_{US} COp$ is verified by a retrieve relation R_1 and that of $AOp \sqsubseteq_{US} X$ by a retrieve relation R_2. Let us denote the state space of COp by C_1 and that of X by C_2. We abbreviate $Astate$ to A.

We first consider applicability. We know that

$$\forall\, C_2 \bullet (\forall\, A \bullet R_2 \implies \text{pre}\, AOp) \implies \text{pre}\, X \qquad (\alpha)$$
$$\forall\, C_1 \bullet (\forall\, A \bullet R_1 \implies \text{pre}\, AOp) \implies \text{pre}\, COp \qquad (\beta)$$

and we need to show that for some retrieve relation R

$$\forall\, C_2 \bullet (\forall\, C_1 \bullet R \implies \text{pre}\, COp) \implies \text{pre}\, X$$

First let us define R as $\exists\, A \bullet R_1 \wedge R_2$. Now suppose that for a given concrete state C_2, $(\forall\, C_1 \bullet R \implies \text{pre}\, COp)$ holds. First note that if C_2 is not in the domain of R_2, then by α, $\text{pre}\, X$ holds at that state. Next suppose that $C_2 \in \text{dom}\, R_2$ and $(C_2, C_1) \notin R$, and consider a state A then $(C_2, A) \in R_2$ implies that $(A, C_1) \notin R_1$. Then by β, α and the definition of COp, $\text{pre}\, X$ holds at state C_2. The case when $(C_2, C_1) \in R$ is similar.

To show correctness holds, we have to show that (see figure below)

$$\forall\, C_2 \bullet (\forall\, C_1 \bullet R \implies \text{pre}\, COp) \implies \forall\, C_1';\ C_2' \bullet (X \wedge R' \implies \exists\, C_1 \bullet R \wedge COp)$$

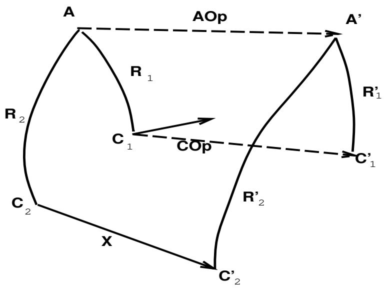
given that we know

$$\forall\, C_2 \bullet (\forall\, A \bullet R_2 \implies \text{pre}\, AOp) \implies \forall\, A';\ C_2' \bullet (X \wedge R_2' \implies \exists\, A \bullet R_2 \wedge AOp)$$

Given that $(\forall\, C_1 \bullet R \implies \text{pre}\, COp)$ implies that $(\forall\, A \bullet R_2 \implies \text{pre}\, AOp)$, by correctness of $AOp \sqsubseteq_{US} X$ we have

$$\forall\, A';\ C_2' \bullet (X \wedge R_2' \implies \exists\, A \bullet R_2 \wedge AOp)$$

Now suppose that given any C_1'; C_2', $X \wedge R'$ implies that $\exists\, C_1 \bullet R \wedge COp$. Now if $X \wedge R'$ then there exists A' with $(C_2, C_2') \in X$, $(C_2', A') \in R_2$, $(A', C_1') \in R_1$. Thus there exists A with $AOp \wedge R_2$. By definition of COp there exists C_1 with $(C_1, C_1') \in COp$ and $(A, C_1) \in R$. That is $\exists\, C_1 \bullet R \wedge COp$ as required.



□

5.2 Generating Tests

The preceding theorem means that to generate concrete tests from the abstract test cases $\{AOp_i\}_i$ we can use the formula

$$COp_i \mathrel{\widehat=} \\ (\forall\, Astate \bullet Ret \implies \text{pre}\, AOp_i) \land \forall\, Astate' \bullet (Ret' \implies \exists\, Astate \bullet Ret \land AOp_i)$$

Since we know that Ret defines a refinement (no need to check applicability), each COp_i is a refinement of AOp_i.

Example 4. Calculating concrete tests from an upward simulation.

Considering the Kurbel specification as an upward simulation of the Marlowe specification we can generate test cases for the Kurbel operations from the abstract test cases of the Marlowe operations. Considered in this direction the retrieve relation is not functional, so we have to use the general formulae given above.

Calculation shows that the abstract *MArrive* test case produces a number of concrete test cases $\{KA_t\}_t$, one for each $t \in kpool$. Similarly, we can calculate concrete tests for the book operation via its test cases $\{MB_t\}_t$, upon refinement these produce one concrete test *KBook* for the Kurbel specification. □

Having shown how to calculate tests we now consider their properties of coverage and disjointness in turn.

We begin with coverage, where we have the following result.

Theorem 5. *Let AOp be an abstract operation with $AOp = \bigvee_i AOp_i$ being its disjoint set of tests. Let COp be an upward simulation of AOp. Let Ret be the retrieve relation. Let COp_i be the concrete tests given above. Then*

$$\bigvee_i COp_i \sqsubseteq_{US} COp$$

and if COp is the weakest upward simulation of AOp then $COp = \bigvee_i COp_i$.

Proof
Let us first observe the following:

$$
\begin{array}{l}
\bigvee_i COp_i \\
= \quad \bigvee_i((\forall Astate \bullet Ret \implies \mathrm{pre}\, AOp_i) \land \\
\qquad\qquad\qquad \forall Astate' \bullet (Ret' \implies \exists Astate \bullet Ret \land AOp_i)) \\
\implies \bigvee_i(\forall Astate \bullet Ret \implies \mathrm{pre}\, AOp_i) \land \\
\qquad\qquad\qquad \bigvee_i(\forall Astate' \bullet (Ret' \implies \exists Astate \bullet Ret \land AOp_i)) \\
\implies (\forall Astate \bullet \bigvee_i(Ret \implies \mathrm{pre}\, AOp_i)) \land \\
\qquad\qquad\qquad (\forall Astate' \bullet \bigvee_i(Ret' \implies \exists Astate \bullet Ret \land AOp_i)) \\
= \quad (\forall Astate \bullet (Ret \implies \bigvee_i \mathrm{pre}\, AOp_i)) \land \\
\qquad\qquad\qquad (\forall Astate' \bullet Ret' \implies \bigvee_i(\exists Astate \bullet Ret \land AOp_i)) \\
= \quad (\forall Astate \bullet (Ret \implies \mathrm{pre} \bigvee_i AOp_i)) \land \\
\qquad\qquad\qquad (\forall Astate' \bullet Ret' \implies (\exists Astate \bullet Ret \land \bigvee_i AOp_i)) \\
\sqsubseteq_{US} COp
\end{array}
$$

Therefore

$$\bigvee_i COp_i \sqsubseteq_{US} COp$$

If COp was in fact the weakest refinement of AOp then we need to show that equality holds between COp and $\bigvee_i COp_i$. This will follow from the fact that

$$
\begin{array}{l}
(\forall Astate \bullet (Ret \implies \mathrm{pre} \bigvee_i AOp_i)) \land \\
\qquad (\forall Astate' \bullet Ret' \implies (\exists Astate \bullet Ret \land \bigvee_i AOp_i)) \\
\qquad\qquad \sqsubseteq_{US} \bigvee_i COp_i
\end{array}
$$

which is easily shown. □

Therefore the covering properties for upward simulations are the same as for downward simulations.

The disjointness properties are also pleasingly symmetric. When the retrieve relation is a surjective function, the formulae for calculating tests is the same as for downward simulations. Therefore, as was the case then, disjoint abstract disjoint tests will produce disjoint concrete tests. However, in general we again find that refined tests are not disjoint.

Example 5. Refined tests are not disjoint in general.

To see this it suffices to consider again the refinement of the Marlowe specification. The retrieve relation is not functional, since the predicates in Ret do not define the abstract space uniquely (in particular, $kpool = mpool \cup \mathrm{ran}\, tkt$ allows many choices of $mpool$ and tkt for a given $kpool$).

Each abstract test (MB_t) of $MBook$ (and there are $\#mpool$ of them) is mapped onto the same concrete test ($KBook$). Therefore the refined concrete tests are not disjoint whereas the abstract ones were. □

6 Conclusions

We have provided a means to calculate concrete tests from abstract ones for both upward and downward simulations. For retrieve relations which are surjective functions the calculations simplified considerably, and in this case the formulae for upward and downward simulations coincide.

We can use this as a basis for a methodology to determine the correct concrete test calculation. Given abstract and concrete state spaces, a retrieve relation and an abstract operation, we proceed as follows:

1. Determine whether *Ret* is a surjective function. If it is, then the concrete tests are given by

   $$COp_i \mathrel{\widehat{=}} \exists\, Astate;\ Astate' \bullet Ret \land AOp_i \land Ret'$$

2. If *Ret* is not a surjective function we determine whether it defines a downward or upward simulation. We do this by determining if

   $$\text{pre}\, AOp \land Ret \Rightarrow \text{pre}\, COp$$

   If this is the case, then the refinement is a downward simulation, and therefore the concrete tests are still given by

   $$COp_i \mathrel{\widehat{=}} \exists\, Astate;\ Astate' \bullet Ret \land AOp_i \land Ret'$$

3. If *Ret* does not define a downward simulation, then the refinement must be an upward simulation. In this case the concrete tests are given by

   $$\begin{aligned} COp_i \mathrel{\widehat{=}} (\forall\, Astate \bullet Ret \implies \text{pre}\, AOp_i) \land \\ \forall\, Astate' \bullet (Ret' \implies \exists\, Astate \bullet Ret \land AOp_i) \end{aligned}$$

4. In all cases, check whether COp was in fact the weakest refinement, we do this by determining if

   $$\bigvee_i COp_i = COp$$

   If this is the case then the set of covering test cases is $\{COp_i\}_i$, if not we may wish to restrict the set of concrete tests further by taking the tests to be $\{COp_i \land COp\}_i$.

Since refining AOp might weaken its precondition, note that it may be necessary to perform further partition analysis in order to place the concrete tests into DNF.

If COp is the weakest refinement of AOp then the set of tests $\{COp_i\}_i$ cover COp. If *Ret* is functional then the concrete tests will be disjoint whenever the abstract tests are disjoint.

In this paper we have just considered the partition analysis for the individual operations to produce a number of test cases derived by conversion of an operation into disjunctive normal form. Further work on this methodology would also consider the partition analysis of the system state and the scheduling of tests to see how these change under refinement.

The partition analysis of the system state again transforms the state into a disjunctive normal form, which is then used to construct a finite state automaton from the specification. The state space changes under refinement and a new partition will be obtained for the concrete state space. We would expect that refinements have a similar effect on the state space to those found for the partition analysis of the operations. This needs to be confirmed.

In addition, we would like to determine whether we can use the retrieve relation to calculate a new FSA for the concrete specification from the abstract one using similar techniques to those above. The scheduling of tests for the concrete specification, which involves finding paths through the FSA which cover all the required tests, would also have to be investigated in light of our discussion of refinement.

References

[1] J. R. Abrial. *The B-Book: Assigning programs to meanings.* CUP, 1996.

[2] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing and Verification, VIII*, pages 63–74, Atlantic City, USA, June 1988. North-Holland.

[3] E. Brinksma, G. Scollo, and C. Steenbergen. Process specification, their implementation and their tests. In B. Sarikaya and G. v. Bochmann, editors, *Protocol Specification, Testing and Verification, VI*, pages 349–360, Montreal, Canada, June 1986. North-Holland.

[4] D. Carrington and P. Stocks. A tale of two paradigms: Formal methods and software testing. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 51–68, Cambridge, United Kingdom, June 1994.

[5] E. Cusack and C. Wezeman. Deriving tests for objects specified in Z. In J. P. Bowen and J. E. Nicholls, editors, *Seventh Annual Z User Workshop*, pages 180–195, London, December 1992. Springer-Verlag.

[6] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(3):83–133, 1984.

[7] J. Derrick and E.A. Boiten. Calculating and verifying refinements of state based specifications. 1998. Submitted of publication.

[8] 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, *FME'93: Industrial-Strength Formal Methods*, pages 268–284. Formal Methods Europe, Springer-Verlag, April 1993. Lecture Notes in Computer Science 670.

[9] J. He. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.

[10] L. Heerink and J. Tretmans. Refusal testing for classes of transition systems with inputs and outputs. In T. Mizuno, N. Shiratori, T. Higashino, and A. Togashi, editors, *FORTE/PSTV XVII'97*. Chapman and Hall, November 1997.

[11] H-M. Horcher. Improving software tests using Z specifications. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 152–166, Limerick, September 1995. Springer-Verlag.

[12] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1989.

[13] M. B. Josephs. The data refinement calculator for Z specifications. *Information Processing Letters*, 27:29–33, February 1988.

[14] G.T. Scullard. Test case selection using VDM. In *VDM '88 VDM – The Way Ahead*, pages 178–186, September 1988.

[15] H. Singh, M. Conrad, and S. Sadeghipour. Test case design based on Z and the classification-tree method. In M. Hinchey and Shaoying Liu, editors, *First IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 81–90, Hiroshima, Japan, November 1997. IEEE Computer Society.

[16] G. Smith and J. Derrick. Refinement and verification of concurrent systems specified in Object-Z and CSP. In M. Hinchey and Shaoying Liu, editors, *First IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 293–302, Hiroshima, Japan, November 1997. IEEE Computer Society.

[17] J. M. Spivey. *The Z notation: A reference manual*. Prentice Hall, 1989.

[18] S. Stepney. Testing as Abstraction. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 137–151, Limerick, September 1995. Springer-Verlag.

[19] P. Stocks and D. Carrington. Deriving software test cases from formal specifications. In *6th Australian Software Engineering Conference*, pages 327–340, July 1991.

[20] L. van Aertryck, M. Benveniste, and D. Le Metayer. Casting: a formally based software test generation method. In M. Hinchey and Shaoying Liu, editors, *First IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 101–110, Hiroshima, Japan, November 1997. IEEE Computer Society.

[21] C. Wezeman and A. J. Judge. Z for managed objects. In J. P. Bowen and J. A. Hall, editors, *Eighth Annual Z User Workshop*, pages 108–119, Cambridge, July 1994. Springer-Verlag.

[22] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.

[23] J. C. P. Woodcock and C. C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C. A. R. Hoare, and H. Langmaack, editors, *VDM '90 VDM and Z - Formal Methods in Software Development*, LNCS 428, pages 340–351, Kiel, FRG, April 1990. Springer-Verlag.