Sharp Retrenchment, Modulated Refinement and Simulation

R. Banach[a], M. Poppleton[a,b]
[a]Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
[b]Faculty of Maths. and Computing, Open University, Milton Keynes, MK7 6AL, U.K.
banach@cs.man.ac.uk , m.r.poppleton@open.ac.uk

Abstract: Sharp retrenchment is introduced and briefly justified informally, as a liberalisation of refinement. In sharp retrenchment the relationship between an abstract operation and its concrete counterpart is mediated by extra predicates, allowing most particularly the description of non-refinement-like properties, and the mixing of I/O and state aspects in the passage between levels of abstraction. Sharp retrenchments are briefly contrasted with unsharp ones. Sharp retrenchments are shown to have a natural law of composition, and the way in which refinements may be viewed as sharp retrenchments is discussed. Modulated refinement is introduced as a version of refinement allowing mixing of I/O and state aspects, in order to facilitate comparison between sharp retrenchment and refinement, and various notions of simulation are considered in this context, specifically: stepwise simulation, the ability of simulator to mimic a sequence of execution steps of the simulatee; strong simulation, in which states and step labels are mapped independently between simulatee and simulator; and the refinement notion itself. Special cases of sharp retrenchment are shown to possess various subsets of these simulation properties, and the extent to which sharp retrenchments contain refinements within them is addressed. The details of the theory are worked out for the B-Method, though the applicability of the underlying ideas is not limited to just that formalism.
Keywords: Retrenchment, Refinement, Simulation, B-Method.

1 Introduction

In [Banach and Poppleton (1998)], retrenchment was proposed as a liberalisation of the notion of refinement, whose purpose was to enable more of the informal aspects of development to be captured within a formal framework. A retrenchment step from a more abstract to a more concrete level of abstraction admits strengthening of the precondition and weakening of the postcondition and the mingling of state and I/O information between the levels, all mediated by two extra predicates per retrenched operation. In particular it allows non-refinement-like behaviour to be expressed via the weakened postcondition; which in turn permits inconvenient low level detail of the true system from cluttering up an idealised model at a higher level of abstraction, a phenomenon which occurs all too frequently when refinement alone is used as the implementation mechanism, as a result of the unforgiving nature of the refinement proof obligations.

[Banach and Poppleton (1998)] was concerned with making the engineering case for the retrenchment notion. We considered it important to proceed in this more pragmatic manner at the outset, as the experience with refinement showed in hindsight that too early an emphasis on mathematical elegance could have detrimental consequences on the development activity for realistic problems. In this paper by contrast, we explore some of the theoretical properties of the retrenchment mechanism, specifically simulation in its various aspects, restricting our attention to notions of simulation in which the granularity of computational steps does not change between levels of abstraction — retrenchment is more aimed at taming the development complexity of individual operations than at increasing potential concurrency. It turns out that there are a number of subtleties regarding simulation to take into account. We take as fundamental the notion of stepwise simulation, in which a sequence of steps of the simulatee is mimicked by an equal length sequence of steps of the simulator. This is closely tied up with the process-algebraic notion of strong simulation — except when the sequence oriented notion does not reproduce in the after-state of a step, the properties used in the before-state, which is what happens in normal refinement. The detailed working out of these issues is the gist of this paper.

Several topics in extant literature can in retrospect be seen as at least partial precursors of the sharp retrenchment of this paper. Perhaps the most obvious one of these is the rely/guarantee method of [Jones (1983)] and its successors. This also uses an additional two predicates per operation, but this time the guarantee predicate is conjunctive to the invariant rather than disjunctive as is the CONCEDES clause of retrenchment, and the emphasis on concurrency lends a different flavour to the enterprise. In fact the sharp retrenchment of this paper also has a third, conjunctive, NEVERTHELESS clause, so we get the best of both worlds. More closely related is the work on clean termination [Coleman and Hughes (1979), Blikle (1981)], dealing with the discrepancy between finite hardware oriented semantic domains and infinite idealised theoretical ones, proof theoretically. This addresses but one of the issues that makes retrenchment useful. On a related tack we have Neilson's thesis [Neilson (1990)], which tackles the same issue by observing that the infinite idealised domains usually arise as well behaved limits of the finite ones, and thus refinement in the idealised case can be understood as the limit of a finite version, giving rise to the notion of acceptably inadequate designs (essentially, an interchange of the order of two quantifiers takes us from idealised to finite). Around the same time [Owe (1985), Owe (1993)] proposes program development using partial functions and a particularly convenient logic, prompted by finiteness and definedness considerations. The ability of retrenchment to mix I/O and state between levels of abstraction, and specifically to change I/O representation was anticipated in [Hayes and Sanders (1995)]. And more recently [Boiten and Derrick (1998), Stepney, Cooper and Woodcock (1998), Mikhajlova and Sekerinski (1997)] treat refinement incorporating change of I/O representation.

A more detailed plan of the rest of this paper is as follows. In Section 2 we briefly discuss why refinement is too strong a notion to conveniently encompass much of realistic system building activity. For a much fuller treatment of such motivational issues see [Banach and Poppleton (1998)]. In Section 3 we introduce sharp retrenchment, the proof obligations that give it semantic content, and give an example. In Section 4 we compare sharp retrenchment with (unsharp) retrenchment as presented in loc. cit.. The more elaborate sharp version turns out to be better suited to discussion of theoretical issues such as those treated in this paper, though we argue that the unsharp version is likely to be quite adequate for most realistic system developments. In Section 5 we summarise some formal facts about the B Method's Abstract Machine Notation and generalised substitutions that provide the mathematical basis for the detailed theory later. In particular we give a notion of "step" in the generalised substitution framework, which enables us to move between automata-theoretic notions of step, and the refinement calculus / weakest precondition notions of step in the world of B. In Section 6 we discuss the composition of sharp retrenchments, and in Section 7 we discuss the extent to which conventional refinement can be viewed as retrenchment. In Section 8 we introduce modulated refinement, a notion which in allowing the mingling of state and I/O information between levels of abstraction in a refinement setting, allows a much better comparison between refinement and retrenchment later. Modulated refinement is introduced in two variants, normal and inverted, depending on the orientation of the dependency between termination predicates in the operation proof obligation. This provides an excellent laboratory for a detailed comparison of different notions of simulation as mentioned above.

In Section 9 we move on to retrenchment proper giving the definition of stepwise simulation for the retrenchment situation. The next three sections deal with the simulation-theoretic and refinement-theoretic properties of some special cases of retrenchment. Section 10 introduces ¬C retrenchment, a notion very close to a modulated refinement, and with correspondingly strong properties. Section 11 deals with ∃GVP retrenchment, a rather weaker notion, with a weaker simulation result, and without an easy refinement theorem. Section 12 deals with decomposed trim retrenchment, a notion designed to yield a refinement theorem, and one that will therefore possess suitable simulation results.

With all of these systems, what happens is in general as one would expect. The more the restrictions imposed make a retrenchment look like a refinement, the more the simulation properties resemble those of modulated refinement. Correspondingly, the more tenuous the link, the less this is the case, highlighting in turn subtle differences between the various notions of simulation introduced. Section 13 comments briefly on variations on the themes in Sections 10-12, possibilities which arise from the relative richness of the retrenchment notation. Section 14 concludes. Table 1 summarises the main concepts introduced in the paper, and the main properties that they enjoy.

Notation. In the body of the paper we use the B Abstract Machine Notation for model oriented specification and system development (see [Abrial (1996a), Wordsworth (1996), Lano and Haughton (1996), Sekerinski and Sere (1998)]). This has the virtue of being a semantically solid formalism that features a satisfying degree of syntactic completeness, including specific concrete syntax for declarations, initialisation, operation specification, and most particularly for refinement. Into this, the ideas of retrenchment can be fitted quite smoothly. Nevertheless, despite this specificity, our proof techniques vary in flavour, from the (in spirit at least) mechanically checkable style of [Abrial (1996a)], to much more model-oriented ones when the latter seem to offer a more intuitively appealing route to the conclusion — Section 5 helps to bridge the gap. Also the main ideas of the paper are largely independent of notation and could quite readily be accomodated into methodologies such as VDM [Jones (1990), Jones and Shaw (1990)] and Z [Spivey (1993), Hayes (1993), Woodcock and Davies (1996)]. To make the paper accessible to otherwise knowledgeable readers unfamiliar with AMN, in the course of the paper we comment on details of notation to an extent that the expert might well regard as superfluous.

2 The Trouble with Refinement

Let us consider a small example. Suppose we must implement the operation of adding an element *new* to a set *xx* ; for the sake of argument let it be a set of natural numbers. In the B notation we would write at the abstract level:

Table 1: Concepts and their Properties

Concept	Properties
Retrenchment	Vertical composition. Subsumes refinement.
Modulated refinement	Normal and inverted forms. Stepwise simulation. Strong simulation for inverted form.
¬C Retrenchment	Stepwise simulation. Strong simulation. Modulated refinement.
∃GVP Retrenchment	Stepwise simulation. Weaker strong simulation.
Decomposed Retrenchment	Modulated refinement. Various stepwise and strong simulation properties.

```
MACHINE            Set_Machine
VARIABLES          xx
INVARIANT          $xx \subseteq \mathsf{NAT}$
INITIALISATION     $xx := \varnothing$
OPERATIONS
    AddElem ( new )  $\triangleq$
        IF
            $new \in \mathsf{NAT}$
        THEN
            $xx := xx \cup \{ new \}$
        END
END
```

We might wish to refine this to a situation in which the set *xx* was represented by an injective sequence of its members, *xx_seq* , initialised empty as was the abstract *xx* , and, for pragmatic reasons limited in length to a maximum of 10 elements. Set union becomes represented by appending sequences. We might attempt to write the resulting refinement as:

```
REFINEMENT         Set_Machine_R
REFINES            Set_Machine
VARIABLES          xx_seq
INVARIANT          $xx\_seq \in \mathrm{iseq}( \mathsf{NAT} ) \wedge \mathrm{size}( xx\_seq ) \leq 10 \wedge$
                   $xx = \mathrm{ran}( xx\_seq )$
INITIALISATION     $xx\_seq := < >$
```

```
OPERATIONS
    $AddElem ( new ) \triangleq$
        IF
            $new \in \mathsf{NAT} \wedge new \notin \mathrm{ran}( xx\_seq ) \wedge \mathrm{size}( xx\_seq ) < 10$
        THEN
            $xx\_seq := xx\_seq \frown [ new ]$
        END
END
```

The trouble is that this is not a refinement in the technical sense. To see this let us examine the proof obligation of refinement:

$$\begin{aligned}&INV_{\mathrm{A}} \wedge INV_{\mathrm{C}} \wedge \mathsf{trm}(AddElem_{\mathrm{A}})\\ &\qquad\Rightarrow \mathsf{trm}(AddElem_{\mathrm{C}}) \wedge [AddElem_{\mathrm{C}}] \neg [AddElem_{\mathrm{A}}] \neg INV_{\mathrm{C}}\end{aligned}$$

Here INV_{A} and INV_{C} are the abstract and concrete invariants respectively (and the concrete invariant contains the retrieve relation $xx = \mathrm{ran}(xx\_seq)$). And $\mathsf{trm}(AddElem_{\mathrm{A}})$ and $\mathsf{trm}(AddElem_{\mathrm{C}})$ are the abstract and concrete termination predicates (normally called preconditions in non-B parlance). Moreover $[AddElem_{\mathrm{C}}] \neg [AddElem_{\mathrm{A}}] \neg INV_{\mathrm{C}}$ asserts that for every step that the concrete generalised substitution (aka operation) $AddElem_{\mathrm{C}}$ makes, there is a step that the abstract generalised substitution $AddElem_{\mathrm{A}}$ can make that establishes the truth of the concrete invariant INV_{C} (which includes the retrieve relation), in the abstract and concrete after-states[1].

Consider what this says when $| xx | = 10$ and $new \notin xx$. We have to show for the first conjunct on the RHS:

$$\begin{aligned}&xx \subseteq \mathsf{NAT} \wedge\\ &xx\_seq \in \mathrm{iseq}(\mathsf{NAT}) \wedge \mathrm{size}(xx\_seq) \leq 10 \wedge xx = \mathrm{ran}(xx\_seq) \wedge \mathsf{true}\\ &\qquad\Rightarrow\\ &\quad \mathsf{true}\end{aligned}$$

which is fine, and asserts that since the abstract system can make a step, the concrete system too can make a step. For the second conjunct we need to show after some working that amongst other things:

$$\begin{aligned}&xx \subseteq \mathsf{NAT} \wedge\\ &xx\_seq \in \mathrm{iseq}(\mathsf{NAT}) \wedge \mathrm{size}(xx\_seq) \leq 10 \wedge xx = \mathrm{ran}(xx\_seq) \wedge \mathsf{true}\\ &\qquad\Rightarrow\\ &\quad new \in \mathsf{NAT} \wedge new \notin xx \wedge \mathrm{size}(xx\_seq) = 10\\ &\qquad\qquad\Rightarrow\\ &\qquad [\mathsf{skip}] \neg [xx := xx \cup \{ new \}] \neg (xx = \mathrm{ran}(xx\_seq))\end{aligned}$$

which is clearly false since in the after-states xx will have an extra element compared to xx_seq . We see that to get a genuine refinement we would have to push the concrete level finiteness requirement up to the abstract level. This would spoil the simplicity and clarity of the abstract system; we would be forced to narrow the gap between the two

1. In B, operations and initialisations are expressed as generalised substitutions, which intuitively work like assignments, and can be nondeterministic. The notation $[S] P$ means that any effect of the generalised substitution S establishes predicate P . Hence the $\forall$–$\exists$–structure above.

levels of abstraction. This in turn weakens the case for having the abstract level there at all.

One way out of this impasse that is much favoured in the formal development community is to amplify the key statement in the abstract operation to:

$xx := xx \cup \{ new \} \; [\,] \; skip$

In this particular case, *skip* , ([ ] is the choice combinator in B), would do just what the concrete operation does in the (null) ELSE branch of the conditional, so a refinement could indeed be recovered. However more generally one would need to write *skip* alone and to have a trivial retrieve relation in order to guarantee a refinement, banking on the fact that in isolation *skip* allows *any* terminating operation (which preserves the retrieve relation) as a refinement. In such a situation the *skip* says nothing at all about the relationship between the "true" abstract and concrete levels, except to signal loud and clear that whatever it is, it is certainly *not* a refinement relationship. As a technique for capturing design decisions in the engineering of real systems in the manner illustrated (as opposed to its proper role as an identity for sequential composition in the calculus of generalised substitutions) we therefore consider *skip* harmful.

We return to our putative refinement above. When we are indeed in the $| xx | = 10$ and $new \notin xx$ situation, the concrete system simply does nothing. Would this be appropriate in reality? Of course not. We would want the operation to at least inform its caller that it was unable to fulfil the normal demand, and that it was taking an exceptional action. For this we would need a change of signature, eg.

$response \longleftarrow AddElem\,(\,new\,)$

where the value of the *response* output would indicate whether a normal or exceptional course of action had been taken. But a change of signature is not permitted in normal refinement, and the presence of the output at the abstract level would be another unnecessary distraction from the essential simplicity of the abstract system. Again, putting in the detail necessary to construct a genuine refinement into the abstract system weakens the case for having the abstract system there at all.

We see that there are drawbacks to using refinement as the sole means of going from an abstract description of a system to a concrete one. Evidently the issues we have raised are rather trivial in the case of such a small illustrative example, but it is not hard to imagine that in realistic situations, the level of detail that needs to be brought up to the abstract system in order for there to be a refinement between abstract and concrete worlds is so great that it overwhelms the underlying concepts of the abstract model. The supposedly abstract model then becomes little more than a restatement of the concrete model in another language. Such a thing is not terrible in itself of course — the different perspectives of the two descriptions can each illuminate the other — but the valuable goal of setting out how the real system definition is arrived at from the designer's original simplified ideas is lost. In this manner we briefly motivate the introduction of a more liberal notion than refinement which we intend to bridge that gap. Such motivational issues are discussed much more extensively in [Banach and Poppleton (1998)].

3 Sharp Retrenchment

Let us refer for a moment to the syntactic details of refinement in B . The top level system construct in B is the MACHINE . Refinement is expressed via another top level construct, the REFINEMENT . A REFINEMENT contains roughly speaking the same ingredients as a MACHINE except that the INVARIANT contains the retrieve relation, and there is the REFINES clause which indicates which other top level construct (either a MACHINE or a previous REFINEMENT) is being refined. We are going to introduce a variant of refinement, sharp retrenchment, but since the result of the development step is in effect the specification of a fresh problem, the resulting top level construct will itself be a MACHINE . For flexibility we permit either a MACHINE or a REFINEMENT to be retrenched. Without further ado we give a schematic sharp retrenchment step.

```
MACHINE          M ( a )          MACHINE          N ( b )
                                  RETRENCHES       M
VARIABLES        u                VARIABLES        v
INVARIANT        I ( u )          INVARIANT        J ( v )
                                  RETRIEVES        G ( u , v )
INITIALISATION   X ( u )          INITIALISATION   Y ( v )
OPERATIONS                        OPERATIONS
    o ⟵ OpName ( i ) ≙               p ⟵ OpName ( j ) ≙
        S ( u , i , o )                   BEGIN
END                                           T ( v , j , p )
                                          LVAR
                                              A
                                          WITHIN
                                              P ( i , j , u , v , A )
                                          CONCEDES
                                              C ( u , v , o , p , A )
                                          NEVERTHELESS
                                              V ( u , v , o , p , A )
                                          END
                                  END                              (3.1)
```

On the left we see a MACHINE $M(a)$, parameterised by a , its operations being simply given by a signature $o \longleftarrow OpName(i)$ and a generalised substitution $S(u, i, o)$. On the right we have a MACHINE $N(b)$, together with the RETRENCHES M clause and retrieve relation RETRIEVES $G(u, v)$. (Note that the retrieve relation is given separately, unlike in B refinement where it is incorporated into the concrete INVARIANT .) Furthermore the body of each operation $p \longleftarrow OpName(j)$ is now a ramified generalised substitution, that is to say a generalised substitution $T(v, j, p)$, together with its ramification, the LVAR , WITHIN , CONCEDES , NEVERTHELESS clauses. Each *OpName* of M must appear ramified within N , but there can also be additional operations in N . (In particular, such additional operations need not preserve the retrieve relation, hence need not be refinements of *skip* .) If one strips away the RETRENCHES clause, the RETRIEVES clause, and the ramifications, one ends up with just a normal B MACHINE .

Speaking informally, the ramification of an operation allows us to describe how the concrete operation fails to refine its abstract counterpart. The LVAR A clause, which is optional, allows us to introduce logical variables A that remember before-values of variables and inputs, so that we may refer to them in the after-state if necessary. The scope of the LVAR declaration is the WITHIN , CONCEDES , and NEVERTHELESS clauses. The job of the WITHIN clause is to describe nontrivial relationships between the abstract and concrete before-values of variables u and v , and abstract and concrete input values i and j , and to define values for the logical variables A . It is used to strengthen the precondition as we will see below, and thus may contain any strengthening of the retrieve relation required in the sharp retrenchment step. The purpose of the CONCEDES and NEVERTHELESS clauses is to provide similar waivers for the after-state. So the CONCEDES clause weakens the retrieve relation required in the after-state, and the NEVERTHELESS clause strengthens it. Both can involve both abstract and concrete variables, abstract and concrete outputs, and the logical variables A . We will see all this more clearly in the proof obligations which we now list.

We assume an environment where all the necessary identifiers are defined in terms of basic types. The POs start with the conventional machine POs for M and N . So there are the initialisation POs:

$$[\ X(u)\ ]\ I(u) \tag{3.2}$$

$$[\ Y(v)\ ]\ J(v) \tag{3.3}$$

and the invariant preservation POs:

$$I(u) \wedge \mathsf{trm}(S(u, i, o)) \Rightarrow [\ S(u, i, o)\ ]\ I(u) \tag{3.4}$$

$$J(v) \wedge \mathsf{trm}(T(v, j, p)) \Rightarrow [\ T(v, j, p)\ ]\ J(v) \tag{3.5}$$

Next we have the sharp retrenchment initialisation PO which is just like the corresponding refinement initialisation PO:

$$[\ Y(v)\ ] \neg [\ X(u)\ ] \neg\ G(u, v) \tag{3.6}$$

and finally we have the sharp retrenchment PO for operations which reads:

$$\begin{aligned} &(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(T(v, j, p)) \wedge P(i, j, u, v, A)) \\ &\qquad \Rightarrow \\ &\quad \mathsf{trm}(S(u, i, o)) \wedge [\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg \\ &\qquad\qquad ((G(u, v) \vee C(u, v, o, p, A)) \wedge V(u, v, o, p, A)) \end{aligned} \tag{3.7}$$

In this, as well as the predictable $(I(u) \wedge G(u, v) \wedge J(v))$ in the antecedent, we assume that the termination condition for the concrete operation $\mathsf{trm}(T(v, j, p))$ is strengthened by the WITHIN clause $P(i, j, u, v, A)$. In the consequent, the termination condition for the abstract operation $\mathsf{trm}(S(u, i, o))$ is deduced, and the familiar "$[\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg$" structure establishes in the after-states, the retrieve relation weakened by the CONCEDES clause and the combination strengthened by the NEVERTHELESS clause. We do not embark here on a detailed discussion to justify the shape of the PO. There is an extensive treatment of the corresponding issue in [Banach and Poppleton (1998)] which applies here verbatim, aside from the more intricate structure of the $((G \vee C) \wedge V)$ formula which is discussed more fully in the next section. Also, the discussion at the end of Section 8 below may be regarded as a post hoc justification for the termination aspects of the PO. In this paper we will take the sharp retrenchment PO for operations as axiomatic, and embark on exploring its consequences.

To see sharp retrenchment at work, let us redo our failed refinement step above as a bona fide sharp retrenchment. Disregarding the appropriateness or otherwise of the concrete operation given, we can ramify it as follows, getting:

MACHINE	*Set_Machine_Ret*
RETRENCHES	*Set_Machine*

```
VARIABLES          xx_seq
INVARIANT          xx_seq ∈ iseq( NAT ) ∧ size( xx_seq ) ≤ 10
RETRIEVES          xx = ran( xx_seq )
INITIALISATION     xx_seq := < >
OPERATIONS
    AddElem ( new ) ≙
        BEGIN
            IF
                new ∈ NAT ∧ new ∉ ran( xx_seq ) ∧ size( xx_seq ) < 10
            THEN
                xx_seq := xx_seq ⌒ [ new ]
            END
        LVAR
            LL , XX
        WITHIN
            LL = size( xx_seq ) ∧ XX = xx_seq
        CONCEDES
            LL = 10 ∧ xx_seq = XX
        NEVERTHELESS
            true
        END
END
```

The above illustrates well how the various additional clauses that constitute the ramification help to make precise the relationship between the two levels, even though it is not a refinement. Note that the occurrences of *xx_seq* in the WITHIN clause refer to the before-value while the occurrence in the CONCEDES clause refers to the after-value, necessitating the use of *LL* and *XX* .

4 Unsharp and Sharp Retrenchment

(Unsharp) retrenchment as introduced in [Banach and Poppleton (1998)] is distinguished by the absence of a NEVERTHELESS clause. The corresponding PO for operations therefore just reads:

$$
\begin{aligned}
&(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(T(v, j, p)) \wedge P(i, j, u, v, A)) \\
&\qquad \Rightarrow \\
&\quad \mathsf{trm}(S(u, i, o)) \wedge [\, T(v, j, p) \,] \neg [\, S(u, i, o) \,] \neg \\
&\qquad\qquad (G(u, v) \vee C(u, v, o, p, A))
\end{aligned}
$$

This form does not allow any strengthening of the retrieve relation in the result of the operation, to be formally expressed. Of course one can include the required clause disjunctively in an enlarged CONCEDES clause since $(G \vee C) \wedge V \Rightarrow G \vee (C \vee V)$. Let us look at an example.

```
MACHINE            Machine_0
VARIABLES          aa , bb , cc
INVARIANT          aa ∈ NAT ∧ bb ∈ NAT ∧ cc ∈ NAT
OPERATIONS
    MyPlus ≙  cc := aa + bb ;
END
```

This can be sharply retrenched as follows. (Note that || is the combinator for parallel composition of generalised substitutions, and that in AMN, *FALSE* is a term while *false* is a predicate.)

```
MACHINE           Machine_1 ( MaxNum )
RETRENCHES        Machine_0
CONSTRAINTS       MaxNum ∈ NAT
VARIABLES         aaa
INVARIANT         aaa ∈ NAT
RETRIEVES         aa = aaa
INITIALISATION    aaa := 3
OPERATIONS
    resp , ccc ⟵ MyPlus ( bbb ) ≙
        BEGIN
            IF
                aaa + bbb ≤ MaxNum
            THEN
                ccc := aaa + bbb ||
                resp := TRUE
            ELSE
                ccc := 0 ||
                resp := FALSE
            END
        LVAR
            CC
        WITHIN
            CC = cc ∧ bb = bbb
        CONCEDES
            false
        NEVERTHELESS
            (resp = TRUE ⇔ cc = ccc) ∧
            (resp = FALSE ⇔ (cc = CC ∧ ccc = 0))
        END
END
```

Now without a NEVERTHELESS clause, one could simply write the ramification as:

```
        LVAR
            CC
        WITHIN
            CC = cc ∧ bb = bbb
        CONCEDES
            (resp = TRUE ⇔ cc = ccc) ∧
            (resp = FALSE ⇔ (cc = CC ∧ ccc = 0))
        END
```

In terms of providing a receptacle for the clause "(*resp* = *TRUE* ⇔ *cc* = *ccc*) & (*resp* = *FALSE* ⇔ (*cc* = *CC* & *ccc* = 0))" the latter form is adequate. The difference between the two forms comes in the discharge of the corresponding POs. In the sharp form we are forced to prove that in corresponding post-states

$((aa = aaa) \vee \mathit{false}) \wedge$ "(*resp* = *TRUE* … *ccc* = 0))" which forces the proof of “($resp = \mathit{TRUE} \ldots ccc = 0$))”. In the unsharp form we must merely prove that

$$(aa = aaa) \vee \text{“}(resp = \mathit{TRUE} \ldots ccc = 0))\text{”}$$

holds. For this it is enough to prove ($aa = aaa$) which is rather trivially true. The other branch of the disjunction is equally true of course, and we are at liberty to prove it, or to prove both, but we are not forced to do so. The choice of whether we prove the other branch is thus a meta level matter, rather than part of the formal framework as in the sharp case.

For most practical system design purposes, the present authors conjecture that unsharp retrenchment will be adequate as a framework for documenting design steps and proving that they relate levels of abstraction in the required manner (eg. the example in the previous section). This is especially so when heuristic reasoning forms an important ingredient of the design process as is suggested might occur in [Banach and Poppleton (1998)]; in such situations the finer details of the formal approach can be a little superfluous. However in the construction of systems of the highest criticality, and when one is concerned with the mathematical properties of the retrenchment formalism, the sharp form is more useful as we shall see. For the remainder of this paper we will consider only sharp retrenchment, and will drop the “sharp” qualifier for brevity.

5 The trm , fis , prd , and stp Predicates

In this section we collect together some facts about generalised substitutions that will prove useful at various points in the more technical remainder of the paper. First we recall from [Abrial (1996a)] that any generalised substitution $S(u, i, o)$ has a normal form in terms of predicates $P(u, i)$ and $Q(u, i, u', o)$ such that for any predicate R , $[\ S\ ]\ R$ iff:

$$P(u, i) \wedge (\forall\ u^{\sim}, o^{\sim} \bullet Q(u, i, u^{\sim}, o^{\sim}) \Rightarrow [\ u, o := u^{\sim}, o^{\sim}\ ]\ R) \ \text{ if } \ u^{\sim}, o^{\sim} \setminus R \qquad (5.1)$$

where $u^{\sim}, o^{\sim} \setminus R$ means $u^{\sim}, o^{\sim}$ are non-free in R . In fact (5.1) is obtained by generalising [Abrial (1996a)] — who does not mention I/O — a little, and treating i and o as normal variables except that i is read-only and o is write-only.

From (5.1) it is easy to prove the monotonicity laws:

$$(\forall\ u^{\sim}, o^{\sim} \bullet A \Rightarrow B) \Rightarrow ([\ S\ ]\ A \Rightarrow [\ S\ ]\ B) \qquad (5.2)$$
$$(\forall\ u^{\sim}, o^{\sim} \bullet A \Rightarrow B) \Rightarrow (\neg\ [\ S\ ] \neg\ A \Rightarrow \neg\ [\ S\ ] \neg\ B) \qquad (5.3)$$

The termination predicate[2] for a generalised substitution S is defined by:

$$\mathsf{trm}(S(u, i, o)) \ = \ [\ S\ ]\ \mathit{true} \qquad (5.4)$$

Note that this is universal, in the sense that it says that all possible effects of S establish at least the trivial *true* predicate. From (5.1) we quickly get:

$$\mathsf{trm}(S(u, i, o)) \ \Leftrightarrow \ P(u, i) \qquad (5.5)$$

so we can write $\mathsf{trm}(S)(u, i)$ to emphasise that $\mathsf{trm}(S)$ is only free in u and i . The existential analogue of trm is the feasibility predicate[3] for S :

2. In [Abrial (1996a)] termination is defined as $\mathsf{trm}(S) = [\ S\ ]\ (u = u)$. This alludes to the frame problem, about which we will be careful at the meta level in this paper.

$$\mathsf{fis}(S(u, i, o)) \;=\; \neg\, [\, S\,]\, \neg\, \mathit{true} \tag{5.6}$$

which is also free only in u and i . From (5.1) we get:

$$\mathsf{fis}(S(u, i, o)) \;\Leftrightarrow\; \neg\, P(u, i) \vee (\exists\, u', o \bullet Q(u, i, u', o)) \tag{5.7}$$

This expresses that there is at least *some* terminating outcome of S when it holds. Usually $\mathsf{trm}(S(u, i, o)) \Rightarrow \mathsf{fis}(S(u, i, o))$, but not always. For consider the substitution $S \equiv u :\in \varnothing$, where $:\in$ is nondeterministic assignment to any member of a set; $\mathsf{trm}(S)$ is true, but $\mathsf{fis}(S)$ is false as the empty set does not contain any element to which u could be assigned. (These statements can be proved from the definition $[\, u :\in W\,]\, R \;\Leftrightarrow\; (\forall\, u' \bullet u' \in W \Rightarrow [\, u := u'\,]\, R)$.)

Closely related to $\mathsf{fis}(S)$ is the before-after predicate $\mathsf{prd}(S(u, i, o))$, which we write more commonly as $\mathsf{prd}(S)(u, i, u', o)$ to display clearly its free variables. We generalise [Abrial (1996a)] slightly, defining this by

$$\mathsf{prd}(S)(u, i, u', o) \;=\; [\, o' := o\,]\, \neg\, [\, S\,]\, \neg\, (u' = u \wedge o' = o) \;\text{ if }\; u', o' \setminus S \tag{5.8}$$

From (5.1) this quickly reduces to

$$\mathsf{prd}(S)(u, i, u', o) \;\Leftrightarrow\; \neg\, P(u, i) \vee Q(u, i, u', o) \tag{5.9}$$

so that $\mathsf{prd}(S)$ actually describes the steps that S is able to perform.

In general we will interpret our formulae relationally in this paper. To this end we establish the convention that the machine variables u , i , o , v , j , p , etc. will take values in sets U , I , O , V , J , P respectively etc., i.e. sets named using the uppercase sans serif letter corresponding to the variable name. Superscripted variables such as $u^{\sim}$, u' etc. will also take values in U , while otherwise unqualified subscripted letters such as u_0 , u_0' , $u_1^{\sim}$ etc. will denote elements of U . Relationally interpreted, trm , fis , prd yield pre , dom , rel , defined by:

$$\mathsf{pre}(S) \;=\; \{(u\,, i) \in \mathsf{U} \times \mathsf{I} \mid \mathsf{trm}(S)(u, i)\} \tag{5.10}$$
$$\mathsf{dom}(S) \;=\; \{(u\,, i) \in \mathsf{U} \times \mathsf{I} \mid \mathsf{fis}(S)(u, i)\} \tag{5.11}$$
$$\mathsf{rel}(S) \;=\; \{((u\,, i)\,, (u'\,, o)) \in (\mathsf{U} \times \mathsf{I}) \times (\mathsf{U} \times \mathsf{O}) \mid \mathsf{prd}(S)(u, i, u', o)\} \tag{5.12}$$

Note the presence of $\neg\, P(u, i)$ in $\mathsf{prd}(S)$. For frame reasons, i.e. because $P(u, i)$ says nothing about u', o , it leads to the presence of $(\mathsf{U} \times \mathsf{I} - \mathsf{pre}(S)) \times (\mathsf{U} \times \mathsf{O})$ inside $\mathsf{rel}(S)$. This masks out any capability of the generalised substitution S to yield a sensible output in places where trm does not hold. Thus the only places where the B framework can sensibly describe before-after behaviour are when termination is guaranteed, and this phenomenon prevents the proper description of partial correctness, in which a nondeterministic operation may possibly terminate with an answer, or may possibly not terminate.

To focus better on the actual steps of S , rather than the fictitious ones coming from $\neg\, P(u, i)$, we introduce a predicate $\mathsf{stp}(S)(u, i, u', o)$ by:

$$\mathsf{stp}(S)(u, i, u', o) \;=\; \mathsf{trm}(S)(u, i) \wedge \mathsf{prd}(S)(u, i, u', o) \tag{5.13}$$

From (5.1) we get:

$$\mathsf{stp}(S)(u, i, u', o) \;\Leftrightarrow\; P(u, i) \wedge Q(u, i, u', o) \tag{5.14}$$

3. In [Abrial (1996a)] feasibility is formulated as $\mathsf{fis}(S) = \neg\, [\, S\,]\, \neg\, (u = u)$.

The relational counterpart of stp is srl :

$$\mathsf{srl}(S) \; = \; \{((u\,,\,i)\,,\,(u'\,,\,o)) \in (\mathsf{U} \times \mathsf{I}) \times (\mathsf{U} \times \mathsf{O}) \mid \mathsf{stp}(S)(u,\,i,\,u',\,o)\} \qquad (5.15)$$

and instead of writing $((u_0,\, i_0),\, (u_1,\, o_1)) \in \mathsf{srl}(S)$, we will often find it more convenient to say that u_0 -(i_0, S, o_1)-› u_1 is a step of the relevant machine M .

The standard B framework is that of total correctness as above which we adhere to for specific calculations in this paper; however partial and general correctness are also valid viewpoints for program development; see [de Roever and Engelhardt (1998)] for an extensive comparison of total and partial correctness from both semantic and syntactic perspectives, and [Dunne et al. (1998)] for a general correctness version of B refinement[4]. Such frameworks can support more discriminating stp notions. In this paper we state results concerning stp in terms that at least hint at generalisation to such wider contexts, usually by refering to when the trm property will be explicitly used. The main reason for even considering partial correctness notions here is that the principal semantic notions of interest in this paper concern sequences of steps that have already completed. In such cases, the possibility that there used to be a risk of nontermination at some point is no longer relevant.

Returning to the normal form we see that:

$$\begin{array}{l} P(u,\, i) \wedge (\forall\, u^{\sim},\, o^{\sim} \bullet Q(u,\, i,\, u^{\sim},\, o^{\sim}) \Rightarrow [\; u,\, o := u^{\sim},\, o^{\sim}\;]\, R) \\ \qquad\quad \Leftrightarrow \\ P(u,\, i) \wedge (\forall\, u^{\sim},\, o^{\sim} \bullet P(u,\, i) \wedge Q(u,\, i,\, u^{\sim},\, o^{\sim}) \Rightarrow [\; u,\, o := u^{\sim},\, o^{\sim}\;]\, R) \end{array}$$

so we can derive the useful alternative normal form:

$$\begin{array}{l} [\; S\;]\, R \\ \qquad\quad \Leftrightarrow \\ \mathsf{trm}(S)(u,\, i) \wedge (\forall\, u^{\sim},\, o^{\sim} \bullet \mathsf{stp}(S)(u,\, i,\, u^{\sim},\, o^{\sim}) \Rightarrow [\; u,\, o := u^{\sim},\, o^{\sim}\;]\, R) \\ \hfill \text{if }\; u^{\sim},\, o^{\sim} \setminus R \end{array} \qquad (5.16)$$

We end with a technical result we will use below. It allows more intuitive reasoning in order to establish operation proof obligations.

Theorem 5.1 Let $S(u,\, i,\, o)$ and $T(v,\, j,\, p)$ be generalised substitutions. Then in the standard relational setting, to establish:

$$[\; T(v,\, j,\, p)\;] \neg [\; S(u,\, i,\, o)\;] \neg R(v,\, p,\, u,\, o)$$

it is sufficient to show:

$\mathsf{trm}(T)(v\,,\, j) \wedge \mathsf{trm}(S)(u\,,\, i) \wedge$
(For every step v_0 -(j_0, T, p_1)-› v_1 of T
there is a step u_0 -(i_0, S, o_1)-› u_1 of S ,
such that $R(v_1,\, p_1,\, u_1,\, o_1)$ holds.)

Proof. We simply substitute (5.16) to get:

$$\begin{array}{l} [\; T(v,\, j,\, p)\;] \neg [\; S(u,\, i,\, o)\;] \neg R(v,\, p,\, u,\, o) \\ \qquad\quad \Leftrightarrow \end{array}$$

4. Adding partial correctness notions to a system like B allows one to distinguish between “intrinsic” trm properties of substitutions such as nondivision by zero, and more “ad hoc” assertions added to aid programme development. See also [Behm et al. (1998)].

$$\begin{array}{l}
(\mathsf{trm}(T)(v, j) \wedge (\forall\, v^{\sim}, p^{\sim} \bullet \mathsf{stp}(T)(v, j, v^{\sim}, p^{\sim}) \Rightarrow [\, v, p := v^{\sim}, p^{\sim} \,] \\
\qquad \neg\, (\mathsf{trm}(S)(u, i) \wedge (\, \forall\, u^{\sim}, o^{\sim} \bullet \mathsf{stp}(S)(u, i, u^{\sim}, o^{\sim}) \Rightarrow [\, u, o := u^{\sim}, o^{\sim} \,] \\
\qquad\qquad \neg\, R(v, p, u, o))) \\
\qquad \Leftrightarrow \\
\mathsf{trm}(T)(v, j) \wedge (\forall\, v^{\sim}, p^{\sim} \bullet \mathsf{stp}(T)(v, j, v^{\sim}, p^{\sim}) \Rightarrow \neg\, (\mathsf{trm}(S)(u, i) \wedge \\
\qquad (\forall\, u^{\sim}, o^{\sim} \bullet \mathsf{stp}(S)(u, i, u^{\sim}, o^{\sim}) \Rightarrow [\, v, p := v^{\sim}, p^{\sim} \,][\, u, o := u^{\sim}, o^{\sim} \,] \\
\qquad\qquad \neg\, R(v, p, u, o))) \\
\qquad \Leftrightarrow \\
\mathsf{trm}(T)(v, j) \wedge (\forall\, v^{\sim}, p^{\sim} \bullet \mathsf{stp}(T)(v, j, v^{\sim}, p^{\sim}) \Rightarrow \neg\, (\mathsf{trm}(S)(u, i) \wedge \\
\qquad (\forall\, u^{\sim}, o^{\sim} \bullet \mathsf{stp}(S)(u, i, u^{\sim}, o^{\sim}) \Rightarrow \neg\, R(v^{\sim}, p^{\sim}, u^{\sim}, o^{\sim}))) \\
\qquad \Leftrightarrow \\
\mathsf{trm}(T)(v, j) \wedge (\forall\, v^{\sim}, p^{\sim} \bullet \mathsf{stp}(T)(v, j, v^{\sim}, p^{\sim}) \Rightarrow (\mathsf{trm}(S)(u, i) \Rightarrow \\
\qquad (\exists\, u^{\sim}, o^{\sim} \bullet \mathsf{stp}(S)(u, i, u^{\sim}, o^{\sim}) \wedge R(v^{\sim}, p^{\sim}, u^{\sim}, o^{\sim}))) \\
\qquad \Leftrightarrow \\
\mathsf{trm}(T)(v, j) \wedge (\mathsf{trm}(S)(u, i) \Rightarrow (\forall\, v^{\sim}, p^{\sim} \bullet \mathsf{stp}(T)(v, j, v^{\sim}, p^{\sim}) \Rightarrow \\
\qquad (\exists\, u^{\sim}, o^{\sim} \bullet \mathsf{stp}(S)(u, i, u^{\sim}, o^{\sim}) \wedge R(v^{\sim}, p^{\sim}, u^{\sim}, o^{\sim}))) \qquad\qquad (5.17)
\end{array}$$

Now $(A \wedge B \wedge C)$ implies $(A \wedge (B \Rightarrow C))$, so setting $A \equiv \mathsf{trm}(T)$, $B \equiv \mathsf{trm}(S)$ and $C \equiv (\forall v^{\sim} \ldots)$ in (5.17), and then interpreting the result in the relational framework, leads to the required conclusion. ☺

6 Composition of Retrenchments

In this section we show that retrenchments compose to give retrenchments. So let us take (3.1) as given and suppose we now have the further retrenchment of $N(b)$:

```
MACHINE           N ( b )                MACHINE           O ( c )
RETRENCHES        M                      RETRENCHES        N
VARIABLES         v                      VARIABLES         w
INVARIANT         J ( v )                INVARIANT         K ( w )
RETRIEVES         G ( u , v )            RETRIEVES         H ( v , w )
INITIALISATION    Y ( v )                INITIALISATION    Z ( w )
OPERATIONS                               OPERATIONS
    p ⟵ OpName ( j ) ≙                       q ⟵ OpName ( k ) ≙
        BEGIN                                    BEGIN
            T ( v , j , p )                          U ( w , k , q )
        LVAR                                     LVAR
            A                                        B
        WITHIN                                   WITHIN
            P ( i , j , u , v , A )                  Q ( j , k , v , w , B )
        CONCEDES                                 CONCEDES
            C ( u , v , o , p , A )                  D ( v , w , p , q , B )
        NEVERTHELESS                             NEVERTHELESS
            V ( u , v , o , p , A )                  W ( v , w , p , q , B )
        END                                      END
END                                      END                                  (6.1)
```

There are the usual machine POs for O which we will not discuss. The retrenchment initialisation PO reads:

$$[\, Z(w) \,]\, \neg\, [\, Y(v) \,]\, \neg\, H(v, w) \qquad (6.2)$$

and the retrenchment operation PO is:

$$\begin{aligned}&(J(v) \wedge H(v, w) \wedge K(w)) \wedge (\mathsf{trm}(U(w, k, q)) \wedge Q(j, k, v, w, B))\\ &\qquad\Rightarrow\\ &\quad \mathsf{trm}(T(v, j, p)) \wedge [\, U(w, k, q)\,] \neg [\, T(v, j, p)\,] \neg\\ &\qquad\quad ((H(v, w) \vee D(v, w, p, q, B)) \wedge W(v, w, p, q, B))\end{aligned} \tag{6.3}$$

Now we address the composition, starting with the composed initialisation PO. Suppose $Z(w)$ initialises the variable w to the value $w_0 \in \mathsf{W}$. By (6.2) there is a $v_0 \in \mathsf{V}$ which $Y(v)$ can initialise the variable v to, such that $H(v_0, w_0)$ holds, where we abuse notation by writing $H(v_0, w_0)$ for the value of the predicate $H(v, w)$ in a valuation in which v is mapped to v_0 , and w is mapped to w_0 (we will persist in such notational abuse for the rest of the paper). Moreover by (3.3), $J(v_0)$ holds. Applying the same reasoning to v_0 and (3.6), we deduce the existence of a u_0 such that $G(u_0, v_0)$ holds. Since w_0 was an arbitrary initialisation of w , we deduce that:

$$[\, Z(w)\,] \neg [\, X(u)\,] \neg (\exists\, v \bullet G(u, v) \wedge J(v) \wedge H(v, w)) \tag{6.4}$$

The purpose of including the term $J(v)$ in the composed retrieve relation will become clear when we consider the composed retrenchment operation PO. For this let (3.7) and (6.3) hold. Suppose for some values $u_0 \in \mathsf{U}$, $i_0 \in \mathsf{I}$, $w_0 \in \mathsf{W}$, $k_0 \in \mathsf{K}$, $q_0 \in \mathsf{Q}$, and B_0 , we have:

$$\begin{aligned}&I(u_0) \wedge K(w_0) \wedge \mathsf{trm}(U(w_0, k_0, q_0)) \wedge\\ &\qquad\qquad (\exists\, v, j, A \bullet G(u_0, v) \wedge J(v) \wedge H(v, w_0) \wedge\\ &\qquad\qquad\ P(i_0, j, u_0, v, A) \wedge Q(j, k_0, v, w_0, B_0))\end{aligned} \tag{6.5}$$

Let $v_0 \in \mathsf{V}$, $j_0 \in \mathsf{J}$, and A_0 witness the existential quantification. Then from (6.3) we deduce that $\mathsf{trm}(T(v_0, j_0, p_0))$ holds for a suitable $p_0 \in \mathsf{P}$ [5]. Using this and (3.7), we repeat the argument and get:

$$\mathsf{trm}(S(u_0, i_0, o_0)) \tag{6.6}$$

for a suitable $o_0 \in \mathsf{O}$.

Now (3.7) and (6.3) promise us for each concrete step an abstract step that establishes a particular predicate in the after-states of abstract and concrete variables. Suppose then that w_0 -(k_0, U, q_0)-› w_0' is a step of $U(w_0, k_0, q_0)$, and let v_0 -(j_0, T, p_0)-› v_0' be a corresponding step of $T(v_0, j_0, p_0)$ by (6.3). From this, and the machine invariant PO for N (which contributes the $J(v_0')$) we know that:

$$\begin{aligned}&J(v_0') \wedge ((H(v_0', w_0') \vee D(v_0', w_0', p_0, q_0, B_0)) \wedge\\ &\qquad\quad W(v_0', w_0', p_0, q_0, B_0))\end{aligned} \tag{6.7}$$

holds in the after-state.

Also let u_0 -(i_0, S, o_0)-› u_0' be a step of $S(u_0, i_0, o_0)$ corresponding to v_0 -(j_0, T, p_0)-› v_0' by (3.7). Then this and the machine invariant PO for N again give us:

$$\begin{aligned}&J(v_0') \wedge ((G(u_0', v_0') \vee C(u_0', v_0', o_0, p_0, A_0)) \wedge\\ &\qquad\quad V(u_0', v_0', o_0, p_0, A_0))\end{aligned} \tag{6.8}$$

5. Note that although $T(v_0, j_0, p_0)$ indeed produces a p_0 , $\mathsf{trm}(T(v_0, j_0, p_0))$ is a property of v_0 and j_0 only, in conformance with the formal trm predicate. Similarly for the other trm properties in the current discourse.

in the after-state. Collecting (6.6)-(6.8) we have proved:

$$\begin{array}{l} \mathsf{trm}(S(u_0, i_0, o_0)) \wedge J(v_0') \wedge \\ \qquad ((H(v_0', w_0') \vee D(v_0', w_0', p_0, q_0, B_0)) \wedge \\ \qquad\; W(v_0', w_0', p_0, q_0, B_0)) \wedge \\ \qquad ((G(u_0', v_0') \vee C(u_0', v_0', o_0, p_0, A_0)) \wedge \\ \qquad\; V(u_0', v_0', o_0, p_0, A_0)) \end{array} \qquad (6.9)$$

Noting that u_0 , i_0 , w_0 , k_0 , q_0 , and B_0 were arbitrary, while o_0 , v_0 , j_0 , p_0 , and A_0 were contingent on them, and noting that we have both $\mathsf{trm}(U(w, k, q))$ and $\mathsf{trm}(S(u, i, o))$, allows us by Theorem 5.1 to write:

$$\begin{array}{l} I(u) \wedge K(w) \wedge \mathsf{trm}(U(w, k, q)) \wedge \\ \qquad (\exists\, v , j , A \bullet G(u, v) \wedge J(v) \wedge H(v, w) \wedge \\ \qquad\; P(i, j, u, v, A) \wedge Q(j, k, v, w, B)) \\ \quad \Rightarrow \\ \mathsf{trm}(S(u, i, o)) \wedge [\, U(w, k, q) \,] \neg [\, S(u, i, o) \,] \neg \\ \qquad \{\exists\, v, p, A \bullet J(v) \wedge \\ \qquad\quad ((H(v, w) \vee D(v, w, p, q, B)) \wedge W(v, w, p, q, B)) \wedge \\ \qquad\quad ((G(u, v) \vee C(u, v, o, p, A)) \wedge V(u, v, o, p, A))\} \end{array} \qquad (6.10)$$

Now we rearrange the antecedent a little, rearrange the formula established in the consequent using the distributive laws, weaken the result of that by distributing the existential quantifiers over a conjunction, and weaken the result of that by dropping some $J(v)$ conjuncts and unused existential quantifiers to get[6]:

$$\begin{array}{l} (I(u) \wedge (\exists\, v \bullet G(u, v) \wedge J(v) \wedge H(v, w)) \wedge K(w)) \wedge \\ \qquad \big(\mathsf{trm}(U(w, k, q)) \wedge (\exists\, v, j, A \bullet G(u, v) \wedge J(v) \wedge H(v, w) \wedge \\ \qquad\; P(i, j, u, v, A) \wedge Q(j, k, v, w, B))\big) \\ \quad \Rightarrow \\ \mathsf{trm}(S(u, i, o)) \wedge [\, U(w, k, q) \,] \neg [\, S(u, i, o) \,] \neg \\ \qquad \big(\{(\exists\, v \bullet (G(u, v) \wedge J(v) \wedge H(v, w))) \vee \\ \qquad\quad (\exists\, v, p, A \bullet \\ \qquad\qquad (G(u, v) \wedge D(v, w, p, q, B)) \vee \\ \qquad\qquad (C(u, v, o, p, A) \wedge H(v, w)) \vee \\ \qquad\qquad (C(u, v, o, p, A) \wedge D(v, w, p, q, B))) \\ \qquad\; \} \wedge (\exists\, v, p, A \bullet V(u, v, o, p, A) \wedge W(v, w, p, q, B))\big) \end{array} \qquad (6.11)$$

This is now in the right shape to be interpreted as a retrenchment operation PO. Thus the following definition is sound.

Definition 6.1 Let $M(a)$ be retrenched to $N(b)$ as in (3.1), and let $N(b)$ be retrenched to $O(c)$ as in (6.1). Then the composition of these two retrenchments is the retrenchment:

MACHINE	$O\,(\,c\,)$
RETRENCHES	M
VARIABLES	w
INVARIANT	$K\,(\,w\,)$

6. In deriving this we made use of the monotonicity law $[U(w)]\neg[S(u)]\neg(\exists v \bullet (F_1 \wedge F_2)) \Rightarrow [U(w)]\neg[S(u)]\neg(\exists v \bullet F_1 \wedge \exists v \bullet F_2)$.

```
RETRIEVES          $( \exists v \bullet G(u,v) \wedge J(v) \wedge H(v,w) )$
INITIALISATION     $Z(w)$
OPERATIONS
    $q \longleftarrow OpName(k) \triangleq$
        BEGIN
            $U(w,k,q)$
        LVAR
            $B$
        WITHIN
            $( \exists v, j, A \bullet G(u,v) \wedge J(v) \wedge H(v,w) \wedge$
                $P(i,j,u,v,A) \wedge Q(j,k,v,w,B) )$
        CONCEDES
            $( \exists v, p, A \bullet$
                $( G(u,v) \wedge D(v,w,p,q,B) ) \vee$
                $( C(u,v,o,p,A) \wedge H(v,w) ) \vee$
                $( C(u,v,o,p,A) \wedge D(v,w,p,q,B) ) )$
        NEVERTHLESS
            $( \exists v, p, A \bullet V(u,v,o,p,A) \wedge W(v,w,p,q,B) )$
        END
END                                                              (6.12)
```

We see that including $J(v)$ in the composed retrieve relation is in fact forced by its presence in the antecedents of (3.7) and (6.3). We discarded the $J(v)$ in the composed CONCEDES clause to avoid excessive clutter, and to avoid wasted work on the part of users of composed retrenchments. Specifically, since J holds for all values of v that allow us to contemplate the composition anyway, its inclusion several times in the CONCEDES clause would be tantamount to forcing designers using retrenchment to reprove J unnecessarily several times over. The omission of these $J(v)$ terms leads to a weak form of associativity.

Proposition 6.2 Up to viewing formulae modulo tautology, and up to discarding intermediate machine invariants from the CONCEDES clause, the law of composition of Definition 6.1 is associative.

7 Refinement as Retrenchment

In this section we will examine how refinement can be viewed as a special case of retrenchment. There are two scenarios in which it makes sense to examine this issue. The first is when refinement is viewed as a relation between machines. This is essentially a categorical perspective in which machines are objects and refinements are arrows (up to certain syntactic equivalences), and is very similar to the perspective that originally spawned the idea of retrenchment. The second scenario is the more traditional B perspective, in which the REFINEMENT is emphatically subordinate to a more abstract MACHINE construct. The theories of these two scenarios are subtly different because the latter requires the construction of the refining machine explicitly. Fortunately this is a standard job.

7.1 Refinement as Relation Between Machines

To permit refinement to become a relation between machines we bend the syntax of B refinement a little in this section so that the retrieve relation appears separately, as in retrenchment. For example:

```
MACHINE          M ( a )           MACHINE          N
                                   REFINES          M
VARIABLES        u                 VARIABLES        v
INVARIANT        I ( u )           INVARIANT        J ( v )
                                   RETRIEVES        G ( u , v )
INITIALISATION   X ( u )           INITIALISATION   Y ( v )
OPERATIONS                         OPERATIONS
    o ⟵ OpName ( i ) ≙                 o ⟵ OpName ( i ) ≙
        S ( u , i , o )                    T ( v , i , o )
END                                END                          (7.1)
```

In this scenario we will have the usual machine POs for M and N , and the semantics of refinement is captured by firstly the refinement initialisation PO:

$$[\ Y(v)\ ] \neg [\ X(u)\ ] \neg G(u, v) \tag{7.2}$$

and secondly the refinement PO for operations:

$$\begin{aligned}&(I(u) \land G(u, v) \land J(v)) \land \mathsf{trm}(S(u, i, o))\\ &\qquad \Rightarrow\\ &\qquad \mathsf{trm}(T(v, i, o)) \land [\ T(v, i, o)\ ] \neg [\ S(u, i, o)\ ] \neg G(u, v)\end{aligned} \tag{7.3}$$

While (7.2) is identical to the retrenchment initialisation PO, (7.3) is not. However from (7.3) we can easily derive:

$$\begin{aligned}&(I(u) \land G(u, v) \land J(v)) \land (\mathsf{trm}(T(v, i, o)) \land \mathsf{trm}(S(u, i, o)))\\ &\qquad \Rightarrow\\ &\qquad \mathsf{trm}(S(u, i, o)) \land [\ T(v, i, o)\ ] \neg [\ S(u, i, o)\ ] \neg\\ &\qquad\qquad ((G(u, v) \lor \mathit{false}) \land \mathit{true})\end{aligned} \tag{7.4}$$

from which we deduce:

Proposition 7.1 Let (7.1) be a refinement between machines. Then (7.5) is a retrenchment.

```
MACHINE          M ( a )           MACHINE          N
                                   RETRENCHES       M
VARIABLES        u                 VARIABLES        v
INVARIANT        I ( u )           INVARIANT        J ( v )
                                   RETRIEVES        G ( u , v )
INITIALISATION   X ( u )           INITIALISATION   Y ( v )
OPERATIONS                         OPERATIONS
    o ⟵ OpName ( i ) ≙                 o ⟵ OpName ( i ) ≙
        S ( u , i , o )                    BEGIN
END                                            T ( v , i , o )
                                           WITHIN
                                               trm( S ( u , i , o ) )
                                           CONCEDES
```

```
                    false
                NEVERTHLESS
                    true
                END
            END                                        (7.5)
```

7.2 B Refinement as Retrenchment

In this section we will treat standard B refinement characterised by the REFINEMENT construct, which as [Abrial (1996a)] p. 524ff. puts it, describes an addition or "differential" to a previously given abstract machine. Suppose that we have the following B refinement:

```
MACHINE           M ( a )          REFINEMENT       N
                                   REFINES          M
VARIABLES         u                VARIABLES        v
INVARIANT         I ( u )          INVARIANT        J ( u , v )
INITIALISATION    X ( u )          INITIALISATION   Y ( v )
OPERATIONS                         OPERATIONS
    o ⟵ OpName ( i ) ≙                 o ⟵ OpName ( i ) ≙
        S ( u , i , o )                    T ( v , i , o )
END                                END                          (7.6)
```

Following [Abrial (1996a)], the truth of the usual refinement POs corresponds to the consistency of a machine as follows:

```
MACHINE           N
VARIABLES         v
INVARIANT         ( ∃ u • I ( u ) ∧ J ( u , v ) )
INITIALISATION    Y ( v )
OPERATIONS
    o ⟵ OpName ( i ) ≙
        T^∃S ( v , i , o )
END                                                             (7.7)
```

where $T^{\exists S}$ is the generalised substitution:

```
T^∃S ( v , i , o )   ≡   PRE
                             ( ∃ u • I ( u ) ∧ J ( u , v ) ∧ trm( S ( u , i , o ) ) )
                         THEN
                             T ( v , i , o )
                         END                                    (7.8)
```

for which the termination predicate is:

$$\mathrm{trm}(T^{\exists S}(v, i, o)) \;=\; (\exists\, u \bullet I(u) \wedge J(u, v) \wedge \mathrm{trm}(S(u, i, o))) \wedge \mathrm{trm}(T(v, i, o)) \quad (7.9)$$

Refering to this machine we claim that the following is a valid retrenchment:

```
MACHINE           M ( a )          MACHINE          N
                                   RETRENCHES       M
VARIABLES         u                VARIABLES        v
INVARIANT         I ( u )          INVARIANT        ( ∃ u • I ( u ) ∧
                                                        J ( u , v ) )
```

```
                                          RETRIEVES          $J ( u , v )$
INITIALISATION    $X ( u )$               INITIALISATION     $Y ( v )$
OPERATIONS                                OPERATIONS
   $o \longleftarrow OpName ( i ) \triangleq$        $o \longleftarrow OpName ( i ) \triangleq$
      $S ( u , i , o )$                         BEGIN
END                                             $T^{\exists S} ( v , i , o )$
                                             WITHIN
                                                $\mathsf{trm}( S ( u , i , o ) )$
                                             CONCEDES
                                                $\mathit{false}$
                                             NEVERTHLESS
                                                $\mathit{true}$
                                             END
                                          END                            (7.10)
```

To be sure of this we need to check that the refinement POs for the given refinement imply the retrenchment proof obligations for the stated retrenchment. Mostly this is easy.

Since the underlying machine *N* of the retrenchment is the same as (7.7), we do not need to establish its consistency separately, relying on [Abrial (1996a)]. Likewise the retrenchment initialisation PO is the same as the refinement one, as in the previous section. We turn our attention to the operation PO.

We know that the refinement proof obligation holds:

$$(I(u) \wedge J(u, v)) \wedge \mathsf{trm}(S(u, i, o)) \\ \Rightarrow \mathsf{trm}(T(v, i, o)) \wedge [\ T(v, i, o)\ ] \neg [\ S(u, i, o)\ ] \neg J(u, v) \tag{7.11}$$

Discarding $\mathsf{trm}(T(v, i, o))$ from the consequent, and strengthening the antecedents allows us to write:

$$(I(u) \wedge J(u, v) \wedge (\exists\ u \bullet I(u) \wedge J(u, v))) \wedge \mathsf{trm}(S(u, i, o)) \wedge \\ (\exists\ u \bullet I(u) \wedge J(u, v) \wedge \mathsf{trm}(S(u, i, o))) \wedge \mathsf{trm}(T(v, i, o)) \\ \Rightarrow \mathsf{trm}(S(u, i, o)) \wedge [\ T(v, i, o)\ ] \neg [\ S(u, i, o)\ ] \neg J(u, v) \tag{7.12}$$

or, noting that we can replace *T* by $T^{\exists S}$ in the consequent as the strengthened antecedents ensure that the stronger trm predicate of $T^{\exists S}$ holds, we write:

$$(I(u) \wedge J(u, v) \wedge (\exists\ u \bullet I(u) \wedge J(u, v))) \wedge (\mathsf{trm}(T^{\exists S}(v, i, o)) \wedge \mathsf{trm}(S(u, i, o))) \\ \Rightarrow \\ \mathsf{trm}(S(u, i, o)) \wedge [\ T^{\exists S}(v, i, o)\ ] \neg [\ S(u, i, o)\ ] \neg \\ ((J(u, v) \vee \mathit{false}) \wedge \mathit{true}) \tag{7.13}$$

So we conclude:

Proposition 7.2 Let (7.6) be a B refinement. Then (7.10) is a retrenchment.

8 Modulated Refinement, Stepwise and Strong Simulation

We now embark on studying the converse of the preceding problem, i.e. we ask to what extent can a retrenchment be regarded as providing a refinement? For this we need to enrich our notion of refinement, since retrenchment allows different I/O signatures at the two levels of abstraction. To cope with this we introduce the notion of modulated refinement. For concreteness' sake we propose a B syntax.

```
MACHINE          M ( a )            MACHINE          N ( b )
                                    MODREF           M
VARIABLES        u                  VARIABLES        v
INVARIANT        I ( u )            INVARIANT        J ( v )
                                    RETRIEVES        G ( u , v )
INITIALISATION   X ( u )            INITIALISATION   Y ( v )
OPERATIONS                          OPERATIONS
    o ⟵ OpName ( i ) ≙                  p ⟵ OpName ( j ) ≙
        S ( u , i , o )                     BEGIN
END                                             T ( v , j , p )
                                            WITHIN
                                                P ( i , j , u , v )
                                            NEVERTHELESS
                                                V ( u , v , o , p )
                                            END
                                    END                          (8.1)
```

Note that this captures a picture of modulated refinement as "relation between machines" rather than as "differential". We do not pursue the latter aspect further in this paper; the relevant results can be recovered routinely as was done in the previous section. In fact the above syntax will serve for two slightly different notions of modulated refinement, *normal* and *inverted* to be introduced shortly. The noteworthy points in the syntax are the MODREF keyword and the WITHIN and NEVERTHELESS clauses, the purpose of which is, on a per operation basis, to express how the abstract and concrete inputs and outputs enhance the retrieve relation. This is seen clearly in the proof obligations. For the moment we will study normal modulated refinement. Let the set of operation names of M (and N) be Ops .

Aside from the usual machine POs for M and N , the semantics of normal modulated refinement is captured by the POs. Firstly for initialisation:

$$[\ Y(v)\ ]\ \neg\ [\ X(u)\ ]\ \neg \\ \qquad (G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\ \exists\, i_m \bullet P_m(i_m, j_m, u, v))) \tag{8.2}$$

Next the PO for operations, which for a typical operation reads:

$$(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(S(u, i, o)) \wedge P(i, j, u, v)) \\ \Rightarrow \\ \mathsf{trm}(T(v, j, p)) \wedge [\ T(v, j, p)\ ]\ \neg\ [\ S(u, i, o)\ ]\ \neg \\ \qquad (G(u, v) \wedge V(u, v, o, p)) \tag{8.3}$$

and lastly the operation compatibility PO, which for a typical operation n reads:

$$G(u, v) \wedge V_n(u, v, o_n, p_n) \Rightarrow \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\ \exists\, i_m \bullet P_m(i_m, j_m, u, v)) \tag{8.4}$$

The role of the operation compatibility PO is to ensure that the result of one step cannot prevent any next step purely because of the relationship between abstract and concrete I/Os and states. Normal modulated refinement is intended for situations where a development step mixes I/O and state aspects of a system, but otherwise preserves information content. A special case arises when only the I/O representation is being changed, as in [Hayes and Sanders (1995)]. Unsurprisingly, normal modulated refinement is a special case of retrenchment.

Proposition 8.1 Let (8.1) be a normal modulated refinement. Then M is retrenched to N via retrieve relation $G(u, v)$ provided each operation *OpName* , with signature $o \longleftarrow$ *OpName* (i) , and defined by generalised substitution $S(u, i, o)$ in M , and with signature $p \longleftarrow$ *OpName* (j) , and defined by generalised substitution $T(v, j, p)$ in N , is equipped in N with the ramification WITHIN $\mathsf{trm}(S(u, i, o)) \wedge P(i, j, u, v)$ CONCEDES *false* NEVERTHELESS $V(u, v, o, p)$.

The main reason for studying normal modulated refinement is that it posesses the natural analogue of the simulation property so characteristic of (normal) refinement. First some notation and definitions.

We will write operations of N as $n \equiv (T_n, P_n, V_n)$, setting out n's components, and similarly for M . To discuss simulation formally we will consider execution sequences of steps of N and corresponding ones of M . We write such a typical execution sequence of N as $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-}\!\rangle\ v_1 \text{-}(j_1, m_1, p_2)\text{-}\!\rangle\ v_2 \ldots]$, where the v_r are successive states and the $\text{-}(j_r, m_r, p_{r+1})\text{-}\!\rangle$ specify the input, operation name and output for successive steps. Here steps are defined formally as in Section 5 by the stp predicate and srl relation, and we will usually find it convenient to quote operation names rather than generalised substitutions. Similarly for M . When discussing properties of these sequences, we will write last($\mathcal{T}$) to denote the index of the last state mentioned in $\mathcal{T}$, and we will write $r \in \mathrm{dom}^{\bullet}(\mathcal{T})$ to mean $r \in [0 \ldots \mathrm{last}(\mathcal{T}) - 1]$ if $\mathcal{T}$ is finite, and $r \in \mathsf{NAT}$ otherwise. Similarly for sequences of any type.

Definition 8.2 Let (8.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-}\!\rangle\ v_1 \text{-}(j_1, m_1, p_2)\text{-}\!\rangle\ v_2 \ldots]$ is a concrete execution sequence, and that $\mathcal{S} \equiv [\ u_0 \text{-}(i_0, m_0, o_1)\text{-}\!\rangle\ u_1 \text{-}(i_1, m_1, o_2)\text{-}\!\rangle\ u_2 \ldots]$ is an abstract execution sequence. Then $\mathcal{S}$ is a stepwise simulation of $\mathcal{T}$ iff $G(u_0, v_0)$ holds, $\mathrm{dom}(\mathcal{T}) = \mathrm{dom}(\mathcal{S})$, and for all $r \in \mathrm{dom}^{\bullet}(\mathcal{T})$:

$$\begin{aligned} &G(u_r, v_r) \wedge P_{m_r}(i_r, j_r, u_r, v_r) \wedge \\ &\qquad G(u_{r+1}, v_{r+1}) \wedge V_{m_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}) \end{aligned} \tag{8.5}$$

Definition 8.3 Let (8.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-}\!\rangle\ v_1 \text{-}(j_1, m_1, p_2)\text{-}\!\rangle\ v_2 \ldots]$ is a concrete execution sequence, with sequence of invoked operation names $ms \equiv [\ m_0, m_1 \ldots]$. We define the abstract trmP predicates and associated preP sets (with respect to $\mathcal{T}$) thus, where S_{m_r} is the body of operation m_r in M :

$$\begin{aligned} \mathsf{trmP}_{M,r,r} &= \textit{true} \\ &\ldots\ \ldots\ \ldots \\ \mathsf{trmP}_{M,r,s} &= P_{m_r}(i_r, j_r, u, v_r) \wedge [\ S_{m_r}\ ]\ \mathsf{trmP}_{M,r+1,s} \end{aligned} \tag{8.6}$$

(where $r < s \in \mathrm{dom}(\mathcal{T})$), and for finite $\mathcal{T}$ with $\mathrm{last}(\mathcal{T}) = \mathrm{size}(ms) = z$:

$$\mathsf{preP}_{M,ms} = \{(u_0, i_0, i_1 \ldots i_{z-1}) \in \mathsf{U} \times \mathsf{I}_0 \times \mathsf{I}_1 \ldots \times \mathsf{I}_{z-1} \mid \mathsf{trmP}_{M,0,z}\} \tag{8.7}$$

Note that these objects depend not only on abstract states and abstract inputs, but tacitly also on the concrete states and inputs appearing in $\mathcal{T}$. We can now prove the following.

Theorem 8.4 Let (8.1) describe a normal modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-}\!\rangle\ v_1 \text{-}(j_1, m_1, p_2)\text{-}\!\rangle\ v_2 \ldots]$, with sequence of invoked operation names $ms \equiv [\ m_0, m_1 \ldots\ ]$, be a finite execution sequence of N . Suppose there is a $(u_0, i_0 \ldots) \in \mathsf{preP}_{M,ms}$ such that u_0 also witnesses the initialisation PO (8.2). Then there is a stepwise simulation $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-› } u_1 \text{ -}(i_1, m_1, o_2)\text{-› } u_2 \ldots\ ]$ of $\mathcal{T}$.

Proof. Let $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-› } v_1 \ldots\ ]$ be as given. If $\mathrm{dom}(\mathcal{T}) = \{0\}$, then the hypothesised u_0 is all we need, as $G(u_0, v_0)$ holds. Otherwise we construct a corresponding $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-› } u_1 \ldots\ ]$ by an induction on $\mathrm{dom}^{\bullet}(\mathcal{T})$.

For $r = 0$, we know from the hypotheses that a $(u_0, i_0 \ldots) \in \mathsf{preP}_{M,ms}$ exists such that $G(u_0, v_0)$ holds. Since $\mathsf{trmP}_{M,0,\mathrm{size}(ms)}(u_0, i_0 \ldots) \Rightarrow \mathsf{trm}(S_{m_0})(u_0, i_0)$, we have that $\mathsf{trm}(S_{m_0})(u_0, i_0) \wedge P_{m_0}(i_0, j_0, u_0, v_0) \wedge G(u_0, v_0)$. From the initialisation POs for M and N we know that $I(u_0)$ and $J(v_0)$ hold. So we have the antecedents of the normal modulated refinement PO for operations, which from the step $v_0 \text{ -}(j_0, m_0, p_1)\text{-› } v_1$ of $\mathcal{T}$, yields a step of M , $u_0 \text{ -}(i_0, m_0, o_1)\text{-› } u_1$, such that $G(u_1, v_1) \wedge V_{m_0}(u_1, v_1, o_1, p_1)$ holds. So we have as required:

$$G(u_0, v_0) \wedge P_{m_0}(i_0, j_0, u_0, v_0) \wedge G(u_1, v_1) \wedge V_{m_0}(u_1, v_1, o_1, p_1)$$

Since $(u_0, i_0 \ldots) \in \mathsf{preP}_{M,ms}$ we conclude that there is a $(u_1, i_1 \ldots) \in \mathsf{preP}_{M,\mathrm{tail}(ms)}$, and we also have the machine invariants $I(u_1)$ and $J(v_1)$.

For the inductive step, suppose $\mathcal{S}$ has been constructed as far as the r'th step. Then we have the machine invariants $I(u_r)$ and $J(v_r)$, and we also have[7] $G(u_r, v_r) \wedge (u_r, i_r \ldots) \in \mathsf{preP}_{M,ms\downarrow r}$. This enables us to perform the inductive step as above. ☺

We cannot extend the above strategy to the case of infinite $\mathcal{T}$ as the predicate $\mathsf{trmP}_{M,0,r}$ does not behave well as r grows without bound: not only does the u aspect of the predicate accumulate "at the front" of the predicate, but we also have an unbounded number of input variables to contend with. These problems require extra hypotheses and a different strategy.

Definition 8.5 Let (8.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-› } v_1 \text{ -}(j_1, m_1, p_2)\text{-› } v_2 \ldots\ ]$ is a(n infinite) concrete execution sequence. Let

$$\mathrm{N}_r^{\mathsf{UI}} = |\ \{(u, i) \in \mathsf{U} \times \mathsf{I}_r \mid G(u, v_r) \wedge \mathsf{trm}(S_{m_r})(u, i) \wedge P_{m_r}(i, j_r, u, v_r)\}\ |$$

We say that $\mathcal{T}$ is retrieve bounded iff:

$$\forall\ r \in \mathrm{dom}^{\bullet}(\mathcal{T}) \bullet \mathrm{N}_r^{\mathsf{UI}} < \infty \qquad (8.8)$$

Note that retrieve boundedness is inspired by the same thought as internal continuity in [Jonsson (1991)] and finite invisible nondeterminism in [Abadi and Lamport (1991)].

Theorem 8.6 Let (8.1) describe a normal modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-› } v_1 \text{ -}(j_1, m_1, p_2)\text{-› } v_2 \ldots\ ]$, with sequence of invoked operation names $ms \equiv [\ m_0, m_1 \ldots\ ]$, be an infinite execution sequence of N . Suppose $\mathcal{T}$ is retrieve bounded. Suppose moreover for each r , that there is a $(u_0, i_0 \ldots) \in \mathsf{preP}_{M,ms\uparrow r}$ such that u_0 also witnesses the initialisation PO (8.2). Then there is a stepwise simulation $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-› } u_1 \text{ -}(i_1, m_1, o_2)\text{-› } u_2 \ldots\ ]$ of $\mathcal{T}$.

7. For a sequence ms , $ms\uparrow r$ is the first r elements of ms , and $ms\downarrow r$ is all except the first r elements of ms , where r refers specifically to cardinality rather than to absolute index values for $\mathrm{dom}(ms)$.

Proof. Let $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-›}\ v_1 \ldots\ ]$ be as given. We show there is a corresponding $\mathcal{S} \equiv [\ u_0 \text{-}(i_0, m_0, o_1)\text{-›}\ u_1 \ldots\ ]$ as follows. We know that each finite prefix of $\mathcal{T}$ can be stepwise simulated because of Theorem 8.4. These finite simulations can be arranged into a tree thus: the root is a special node at level –1 ; the nodes at level r are (u, i) pairs such that $G(u, v_r) \wedge \mathsf{trm}(S_{m_r})(u, i) \wedge P_{m_r}(i, j_r, u, v_r)$ holds; and there is an edge of the tree from (u_r, i_r) at level r to (u_{r+1}, i_{r+1}) at level r+1 iff there is a simulation of a finite prefix of $\mathcal{T}$ with $u_r \text{-}(i_r, m_r, o_{r+1})\text{-›}\ u_{r+1}$ as final step (also there are edges from the root to all level 0 nodes). Because there are infinitely many finite simulations the tree is infinite, and by retrieve boundedness each of its levels is finite. By König's Lemma, the tree has an infinite branch, which corresponds to a stepwise simulation $\mathcal{S}$ of $\mathcal{T}$. ☺

Now we introduce inverted modulated refinement. The only difference compared to normal modulated refinement is that instead of (8.3), the operation PO for inverted modulated refinement reads:

$$
\begin{aligned}
&(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(T(v, j, p)) \wedge P(i, j, u, v))\\
&\qquad\Rightarrow\\
&\quad \mathsf{trm}(S(u, i, o)) \wedge [\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg\\
&\qquad\qquad (G(u, v) \wedge V(u, v, o, p))
\end{aligned}
\tag{8.9}
$$

Note the inverted roles of the trm predicates. Evidently:

Proposition 8.7 Let (8.1) be an inverted modulated refinement. Then M is retrenched to N via retrieve relation $G(u, v)$ provided each operation *OpName* , with signature $o \longleftarrow$ *OpName* (i) , and defined by generalised substitution $S(u, i, o)$ in M , and with signature $p \longleftarrow$ *OpName* (j) , and defined by generalised substitution $T(v, j, p)$ in N , is equipped in N with the ramification WITHIN $P(i, j, u, v)$ CONCEDES *false* NEVERTHELESS $V(u, v, o, p)$.

Definition 8.8 Let $\mathcal{S} \equiv [\ u_0 \text{-}(i_0, m_0, o_1)\text{-›}\ u_1 \text{-}(i_1, m_1, o_2)\text{-›}\ u_2 \ldots\ ]$ be an execution sequence of M with sequence of operation names $ms \equiv [\ m_0, m_1 \ldots\ ]$. We say $\mathcal{S}$ is trm-safe iff for all $r \in \mathrm{dom}^{\bullet}(\mathcal{S})$, $\mathsf{trm}(S_{m_r})(u_r, i_r)$ holds.

Note that with our formal definition of execution steps, all execution sequences are trm-safe, but this is not necessarily true in a partial correctness setting. We mention trm-safety below to emphasise when we are going to make use of the termination properties of a hypothesised execution sequence in an essential way.

Theorem 8.9 Let (8.1) describe an inverted modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-›}\ v_1 \text{-}(j_1, m_1, p_2)\text{-›}\ v_2 \ldots\ ]$ be a trm-safe execution sequence of N . Then there is a stepwise simulation $\mathcal{S} \equiv [\ u_0 \text{-}(i_0, m_0, o_1)\text{-›}\ u_1 \text{-}(i_1, m_1, o_2)\text{-›}\ u_2 \ldots\ ]$ of $\mathcal{T}$.

Proof. Let $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-›}\ v_1 \ldots\ ]$ be an execution sequence of N . The $\mathrm{dom}(\mathcal{T}) = \{0\}$ case is as in Theorem 8.4. Otherwise we go by induction on $\mathrm{dom}^{\bullet}(\mathcal{T})$.

For $r = 0$, we know that for the given v_0 and j_0 from $\mathcal{T}$, (8.2) holds. So for the m_0 from $\mathcal{T}$ we can find an i_0 such that $G(u_0, v_0) \wedge P_{m_0}(i_0, j_0, u_0, v_0)$ holds. Now $\mathcal{T}$ is trm-safe so $\mathsf{trm}(T_{m_0})(v_0, j_0)$ holds. Also the initialisation POs for M and N yield $I(u_0)$ and $J(v_0)$. Thus the operation PO (8.9), yields a step of M , $u_0 \text{-}(i_0, m_0, o_1)\text{-›}\ u_1$ such that $G(u_1, v_1) \wedge V_{m_0}(u_1, v_1, o_1, p_1)$ holds. So:

$$G(u_0, v_0) \wedge P_{m_0}(i_0, j_0, u_0, v_0) \wedge G(u_1, v_1) \wedge V_{m_0}(u_1, v_1, o_1, p_1)$$

For the inductive step, suppose S has been constructed as far as the r'th step. Then we have $I(u_r)$, $J(v_r)$, and $G(u_r, v_r) \wedge V_{m_{r-1}}(u_r, v_r, o_r, p_r) \wedge \mathsf{trm}(T_{m_r})(v_r, j_r)$. From the operation compatibility PO (8.4), we can infer that for the j_r and m_r given by T, we can find an i_r such that $P_{m_r}(i_r, j_r, u_r, v_r)$ holds. And this is enough to complete the inductive step as before. ☺

We can see clearly why we need different termination assumptions in Theorems 8.4/6 and 8.9. In the latter, we need a termination assumption about the steps of T, a given execution sequence, to be able to exploit the operation PO. Postulating the individual concrete steps to each be within their individual trm predicates is thus sufficient. In the former, we need a termination assumption about the steps of S, an execution sequence yet to be constructed. Therefore stronger assumptions are needed before the relevant operation PO can be used.

Theorem 8.9 can be understood as establishing a strong simulation in the automata theoretic sense, an intrinsically more local concept than the property in Theorem 8.4.

Definition 8.10 Let $\boldsymbol{U}$ and $\boldsymbol{V}$ be sets (of states), and let $\boldsymbol{U}_0 \subseteq \boldsymbol{U}$ and $\boldsymbol{V}_0 \subseteq \boldsymbol{V}$ be subsets (of initial states). Let $\boldsymbol{L}_\mathrm{U}$ and $\boldsymbol{L}_\mathrm{V}$ be sets (of transition labels). Let $\boldsymbol{U}_0$ and $T_\mathrm{U} \subseteq \boldsymbol{U} \times \boldsymbol{L}_\mathrm{U} \times \boldsymbol{U}$ be a transition system on $\boldsymbol{U}$, and $\boldsymbol{V}_0$ and $T_\mathrm{V} \subseteq \boldsymbol{V} \times \boldsymbol{L}_\mathrm{V} \times \boldsymbol{V}$ be a transition system on $\boldsymbol{V}$. A pair of relations

$$(\Theta_\mathrm{S} : \boldsymbol{U} \leftrightarrow \boldsymbol{V}, \Theta_\mathrm{L} : \boldsymbol{L}_\mathrm{U} \leftrightarrow \boldsymbol{L}_\mathrm{V})$$

is called a strong simulation from T_V to T_U iff:

$$v_0 \in \boldsymbol{V}_0 \Rightarrow \exists\, u_0 \in \boldsymbol{U}_0 \bullet u_0\, \Theta_\mathrm{S}\, v_0 \tag{8.10}$$

and for all u, v:

$$\begin{aligned} &u\, \Theta_\mathrm{S}\, v \;\wedge\; v \text{ -}\mu\text{-› } v' \in T_\mathrm{V} \;\Rightarrow \\ &\qquad \exists\, u' \in \boldsymbol{U}, \lambda \in \boldsymbol{L}_\mathrm{U} \bullet u'\, \Theta_\mathrm{S}\, v' \;\wedge\; \lambda\, \Theta_\mathrm{L}\, \mu \;\wedge\; u \text{ -}\lambda\text{-› } u' \in T_\mathrm{U} \end{aligned} \tag{8.11}$$

Note that Definition 8.10 differs slightly from the conventional notion of strong simulation insofar as initial states of N are not required to relate via Θ_S *solely* to initial states of M. Evidently this is for easier comparison with the initialisations arising from refinements and retrenchments.

Definition 8.11 Let M be a machine with operation names set Ops. Then the trm-safe reachable transition system $T^s{}_M$ associated to M is the initial state set $\boldsymbol{U}_0^s$ and subset $T^s{}_M$ of $\boldsymbol{U}^s \times \boldsymbol{L}_M \times \boldsymbol{U}^s$ where:

(1) $\boldsymbol{L}_M = \{(i, m, o) \in \mathsf{I} \times \mathsf{Ops} \times \mathsf{O}\}$
(2) $\boldsymbol{U}^s = \{u \in \mathsf{U} \mid$ there is a trm-safe execution sequence of M with u as final state$\}$
(3) $\boldsymbol{U}_0^s = \{u_0 \in \boldsymbol{U}^s \mid u_0$ is an initial state of $M\}$
(4) $T^s{}_M = \{u \text{ -}(i, m, o)\text{-› } u' \mid$ there is a trm-safe execution sequence of M with $u \text{ -}(i, m, o)\text{-› } u'$ as final step$\}$

Theorem 8.12 Let (8.1) describe an inverted modulated refinement where the common set of operation names is Ops. Then there is a strong simulation from the trm-safe reachable transition system $T^s{}_N$ of N to the trm-safe reachable transition system $T^s{}_M$ of M.

Proof. We define a strong simulation $(\Theta_\mathrm{S}, \Theta_\mathrm{L})$ as follows:

$$\Theta_\mathrm{S} = \{(u, v) \in \boldsymbol{U}^s \times \boldsymbol{V}^s \mid G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\, \exists\, i_m \bullet P_m(i_m, j_m, u, v))\} \tag{8.12}$$

$$\Theta_L = \{((i, m, o), (j, m, p)) \in (\mathsf{I} \times \mathsf{Ops} \times \mathsf{O}) \times (\mathsf{J} \times \mathsf{Ops} \times \mathsf{P}) \mid (\exists\, u, v \bullet P_m(i, j, u, v)) \wedge (\exists\, u, v \bullet V_m(u, v, o, p))\} \qquad (8.13)$$

Showing that this constitutes a strong simulation is a matter of routinely reworking the details of the proof of Theorem 8.9. ☺

We note that a strong simulation relates states to states, and transition labels to transition labels, essentially independently. In the case of Theorem 8.4, the properties demanded of abstract states at the beginning and end of a transition u -(i, m, o)-› u' , specifically that $(u, i \ldots) \in \mathsf{preP}_{M,m\rightarrow ms}$ and $(u' \ldots) \in \mathsf{preP}_{M,ms}$, depend on m and ms ; in particular a step of M is not guaranteed to reestablish in the after-state, the condition assumed in the before-state, a prerequisite for the successful formulation of separate relations Θ_S and Θ_L . At least this is the case for the notion of (automata theoretic) state that we are working with in this paper, i.e. where the automata theoretic state is the same state that appears in the machines of interest.

There are other possibilities. If we permitted the incorporation of history information in our notion of automata theoretic state (as is done in eg. [Abadi and Lamport (1991), Jonsson (1989, 1991), Lynch (1989), Merritt (1989)]), a notion of strong simulation could be set up for Theorem 8.4, but the identity of notion of state that we are trying to maintain would be lost. So there is no precise analogue of Theorem 8.12 for this case.

The preceding results have set out a path from normal modulated refinement which yields a conventional notion of refinement, to inverted modulated refinement which yields a conventional notion of strong simulation. All these notions generate preorders on machines; for normal refinement and strong simulation the preorder properties are essentially standard results, and for inverted refinement the preorder property can alternatively be infered fom Section 6.

Which notion is most appropriate at any juncture depends very much on circumstances. Conventional refinement is used when we care that the concrete system can provide an implementation at any point that the abstract system can define a step, but we don't care how any abstract nondeterminism is resolved — hence the appearance of the abstract trm predicate in the antecedents, and the relative order of the quantifiers in the step relation in the operation PO. By contrast, the inverted viewpoint is more appropriate when we wish to view the abstract system as a guide to understanding the concrete system — here we do not insist that the concrete system implements the abstract one. Instead it is the places where the concrete system is able to provide a step that we wish to relate to the abstract world. Hence the position of the concrete trm predicate in the inverted operation PO, and the same relative order of the quantifiers in the step relation in the operation PO. Still it would be fair to say that the inverted viewpoint really comes into its own only in the more widely drawn arena of retrenchment.

It is interesting to note that only the inverted case supports a "proper" notion of simulation. In contrast, in normal refinement, the roles of "simulator" and "simulatee" are exquisitely confused; the abstract system says (via the trm predicate) *when* a step must exist and demands that the concrete system complies, while the concrete system says (via the step relation) *how* a step can be performed and demands that the abstract system simulates it. It is noteworthy that many papers on refinement speak casually about its simulation properties but without exploring this point fully. Since refinement is (almost) invariably discussed in a single step context, the sequence-dependent subtleties easily escape attention.

9 Retrenchment and Stepwise Simulation

For the rest of the paper we investigate the simulation and modulated refinement properties of the full retrenchment framework. In general we need to distinguish Ops^M , the operation names at the abstract level, from Ops^N the operation names at the concrete level, where $\mathsf{Ops}^M \subseteq \mathsf{Ops}^N$. First we say what we mean by stepwise simulation in this wider setting.

Definition 9.1 Let (3.1) be a retrenchment. Suppose that $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-›}\ v_1 \text{-}(j_1, m_1, p_2)\text{-›}\ v_2 \ldots\ ]$ is an execution sequence of N , and that $\mathcal{S} \equiv [\ u_0 \text{-}(i_0, m_0, o_1)\text{-›}\ u_1 \text{-}(i_1, m_1, o_2)\text{-›}\ u_2 \ldots\ ]$ is an execution sequence of M , where $[\ m_0, m_1, \ldots\ ]$ is a sequence over Ops^M . Then $\mathcal{S}$ is a stepwise simulation of $\mathcal{T}$ iff $G(u_0, v_0)$ holds, and for all $r \in \mathrm{dom}^\bullet(\mathcal{T})$ there is an A_r such that:

$$\begin{aligned} & G(u_r, v_r) \wedge P_{m_r}(i_r, j_r, u_r, v_r, A_r) \wedge \\ & \qquad (G(u_{r+1}, v_{r+1}) \vee C_{m_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}, A_r)) \wedge \\ & \qquad V_{m_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}, A_r) \end{aligned} \tag{9.1}$$

Because of the flexibility of the retrenchment notion it is clearly futile to expect that an arbitrary retrenchment will yield either a modulated refinement, or a stepwise simulation for an arbitrarily chosen concrete execution sequence. The next three sections are devoted to examining various sets of sufficient conditions, which when imposed on top of the basic retrenchment framework, cause some or all of these or similar properties to hold.

10 ¬C Retrenchment

The basic idea of ¬C retrenchment is that the NEVERTHLESS clause is strong enough to deny the potentially deleterious effects for stepwise simulation of the CONCEDES clause (and, by the way, to ensure that the WITHIN clauses of subsequent operations are satisfiable). As a consequence, almost all the characteristic phenomena of retrenchment are suppressed, and ¬C retrenchments are almost indistinguishable from modulated refinements. This makes them less useful in practical applications, but their extreme resemblance to modulated refinements ensures that a rich collection of theoretical properties can be demonstrated in a relatively general way. If nothing else, these provide a reference point against which weaker special cases can be compared.

Definition 10.1 For a retrenchment like (3.1), suppose the joint initialisation establishes:

$$(G(u_0, v_0) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m\ \exists\, i_m, A_m \bullet P_m(i_m, j_m, u_0, v_0, A_m))) \tag{10.1}$$

and suppose that each Ops^M operation $n \equiv (T_n, A_n, P_n, C_n, V_n)$ of N satisfies the operation compatibility PO:

$$\begin{aligned} & V_n(u, v, o, p, B) \\ & \qquad \Rightarrow \\ & \quad (\neg C_n(u, v, o, p, B) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m\ \exists\, i_m, A_m \bullet P_m(i_m, j_m, u, v, A_m))) \end{aligned} \tag{10.2}$$

then we say that the retrenchment is a ¬C retrenchment.

Theorem 10.2 Let (3.1) describe a ¬C retrenchment where the set of abstract operation names is Ops^M . Let $\mathcal{T} \equiv [\ v_0 \text{-}(j_0, m_0, p_1)\text{-›}\ v_1 \text{-}(j_1, m_1, p_2)\text{-›}\ v_2 \ldots\ ]$ be a trm-safe execution sequence of N . Suppose that the sequence of invoked operation names $ms \equiv$

[m_0, m_1 …] is an Ops^M sequence. Then there is a stepwise simulation $\mathcal{S} \equiv$ [u_0 -(i_0, m_0, o_1)-› u_1 -(i_1, m_1, o_2)-› u_2 …] of $\mathcal{T}$.

Proof. This is very similar to Theorem 8.9. We redo just the heart of the inductive step. So suppose $\mathcal{S}$ has been constructed as far as the r'th step so that we have $(G(u_r, v_r) \vee C_{m_{r-1}}(u_r, v_r, o_r, p_r, A_{r-1})) \wedge V_{m_{r-1}}(u_r, v_r, o_r, p_r, A_{r-1})$. Since we have $V_{m_{r-1}}(u_r, v_r, o_r, p_r, A_{r-1})$, (10.2) permits us to infer that for the j_r and m_r given by $\mathcal{T}$, we can find an i_r and A_r such that $\neg C_{m_{r-1}}(u_r, v_r, o_r, p_r, A_{r-1}) \wedge P_{m_r}(i_r, j_r, u_r, v_r, A_r)$ holds. Conjoining these, then simplifying and weakening the result, shows that $G(u_r, v_r) \wedge P_{m_r}(i_r, j_r, u_r, v_r, A_r)$ is true. This is enough to complete the inductive step as before. ☺

In order to bring out the strong simulation result analogous to Theorem 8.12 for ¬C retrenchment we need a little more notation.

Definition 10.3 Let M and N be machines with operation name sets Ops^M and Ops^N . The M-restricted trm-safe reachable transition system of N (whose states are solely those reachable by trm-safe sequences of operations with names in Ops^M) is defined by setting:

(1) $\boldsymbol{L}_{N_M} = \{(j, m, p) \in \mathsf{J} \times \mathsf{Ops}^M \times \mathsf{P}\}$
(2) $\boldsymbol{V}^s_M = \{v \in \mathsf{V} \mid$ there is a trm-safe execution sequence of N consisting solely of Ops^M operations, with v as final state$\}$
(3) $\boldsymbol{V}^s_{M0} = \{v_0 \in \boldsymbol{V}^s_M \mid v_0$ is an initial state of $N\}$
(4) $T^s_{N_M} = \{v$ -(j, m, p)-› $v' \mid$ there is a trm-safe execution sequence of N consisting solely of Ops^M operations, with v -(j, m, p)-› v' as final step$\}$

Theorem 10.4 Let (3.1) describe a ¬C retrenchment where the set of abstract operation names is Ops^M . Then there is a strong simulation from the M-restricted trm-safe reachable transition system $T^s_{N_M}$ of N to the trm-safe reachable transition system T^s_M of M .

Proof. We define a strong simulation (Θ_S, Θ_L) as follows, after which the details are routine.

$$\Theta_S = \{(u, v) \in \boldsymbol{U}^s \times \boldsymbol{V}^s_M \mid G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall j_m \exists i_m, A_m \bullet P_m(i_m, j_m, u, v, A_m))\}$$
$$\Theta_L = \{((i, m, o), (j, m, p)) \in (\mathsf{I} \times \mathsf{Ops}^M \times \mathsf{O}) \times (\mathsf{J} \times \mathsf{Ops}^M \times \mathsf{P}) \mid (\exists A \bullet (\exists u, v \bullet P_m(i, j, u, v, A)) \wedge (\exists u, v \bullet V_m(u, v, o, p, A)))\}$$

☺

¬C retrenchment is sufficiently strong to yield a notion of modulated refinement. Here and in similar results below we disregard any operation of N not in Ops^M of course. We present a number of closely related results. For the first of these we give a full proof, the other cases following easily.

Theorem 10.5 Let (3.1) describe a ¬C retrenchment where the set of abstract operation names is Ops^M . Then the following is a normal modulated refinement:

MACHINE	$M^{\forall T}$ (a)	MACHINE	$N^{\exists A}$ (b)
		MODREF	$M^{\forall T}$
VARIABLES	u	VARIABLES	v
INVARIANT	I (u)	INVARIANT	J (v)
		RETRIEVES	G (u , v)

```
INITIALISATION  $X(u)$                  INITIALISATION  $Y(v)$
OPERATIONS                              OPERATIONS
    $o \longleftarrow OpName(i) \triangleq$            $p \longleftarrow OpName(j) \triangleq$
        $S^{\forall T}(u, i, o)$                    BEGIN
END                                             $T(v, j, p)$
                                            WITHIN
                                                $P^{\exists A}(i, j, u, v)$
                                            NEVERTHELESS
                                                $V^{\exists A}(u, v, o, p)$
                                            END
                                        END                          (10.3)
```

where:

```
$S^{\forall T}(u, i, o) \equiv$ PRE
        $(\forall v, j \bullet I(u) \wedge G(u, v) \wedge J(v) \wedge \mathsf{trm}(S(u, i, o))$
            $\Rightarrow \mathsf{trm}(T(v, j, p)))$
    THEN
        $S(u, i, o)$
    END                                                  (10.4)
```

$$P^{\exists A}(i, j, u, v) \equiv (\exists A \bullet P(i, j, u, v, A)) \tag{10.5}$$

$$V^{\exists A}(u, v, o, p) \equiv (\exists A \bullet V(u, v, o, p, A)) \tag{10.6}$$

Proof. Since $M^{\forall T}$ is a fresh machine, we must check its consistency. The initialisation PO is as for M. And the operation consistency PO, $I \wedge \mathsf{trm}(S^{\forall T}) \Rightarrow [S^{\forall T}]\, I$, follows since[8] for any Φ, $\mathsf{trm}(\Phi|S) \Leftrightarrow \Phi \wedge \mathsf{trm}(S)$, and $[\Phi|S]\, I \Leftrightarrow \Phi \wedge [S]\, I$. For the rest of this proof, let Φ refer specifically to the PRE clause $(\forall v, j \bullet I(u) \ldots \mathsf{trm}(T(v, j, p)))$ introduced in (10.4) above.

We can now check that the POs of the ¬C retrenchment imply those of the refinement. For the initialisation PO, reinterpreting the $\exists A$ in (10.1), yields (8.2) with $P_m^{\exists A}$ replacing the P_m. Likewise for the operation compatibility PO, weakening, and reinterpreting the $\exists A$ in (10.2), yields (8.4) with $P_m^{\exists A}$ again replacing the P_m.

For the operation PO we need to show that for all members of Ops^M :

$$\begin{aligned}&(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(S^{\forall T}(u, i, o)) \wedge P^{\exists A}(i, j, u, v))\\ &\quad\Rightarrow\\ &\quad \mathsf{trm}(T(v, j, p)) \wedge [T(v, j, p)]\, \neg\, [S^{\forall T}(u, i, o)]\, \neg\\ &\qquad\quad (G(u, v) \wedge V^{\exists A}(u, v, o, p))\end{aligned} \tag{10.7}$$

knowing that (3.7) and (10.2) hold. So let us hypothesise the antecedents of (10.7). These contain the ingredients of an application of generalised modus ponens, from which we can infer $\mathsf{trm}(T(v, j, p))$, as required in the consequent of (10.7); also through foresight, we add $\mathsf{trm}(T(v, j, p))$ to the hypotheses. Now since we assume $P^{\exists A}(i, j, u, v)$, we can infer $P(i, j, u, v, A)$ for some A, and we add this $P(i, j, u, v, A)$ to the hypotheses. At this point the hypotheses contain the antecedents of (3.7), so we infer in particular that

8. $\Phi|S$ is the substitution S preconditioned by Φ, so $\Phi|S$ is also PRE Φ THEN S END .

$$[\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg \\ \quad ((G(u, v) \vee C(u, v, o, p, A)) \wedge V(u, v, o, p, A))$$

Next we use the normal form for generalised substitutions from Section 5. Applying this to both T and S , and pushing the v, p substitution to the right we get:

$$\mathsf{trm}(T) \wedge (\forall v^{\sim} p^{\sim} \bullet \mathsf{stp}(T) \Rightarrow \neg\ (\mathsf{trm}(S) \wedge (\forall u^{\sim} o^{\sim} \bullet \mathsf{stp}(S) \Rightarrow \dots \dots)))$$

We lose nothing by disjoining $\neg\Phi$ to the (outer) right conjuct since Φ is one of our hypotheses. After a little simplification we obtain:

$$\mathsf{trm}(T) \wedge (\forall v^{\sim} p^{\sim} \bullet \mathsf{stp}(T) \Rightarrow \neg\ (\Phi \wedge \mathsf{trm}(S) \wedge (\forall u^{\sim} o^{\sim} \bullet \mathsf{stp}(S) \Rightarrow \dots \dots)))$$

whence

$$\mathsf{trm}(T) \wedge (\forall v^{\sim} p^{\sim} \bullet \mathsf{stp}(T) \Rightarrow \neg\ (\mathsf{trm}(\Phi|S) \wedge (\forall u^{\sim} o^{\sim} \bullet \mathsf{stp}(\Phi|S) \Rightarrow \dots \dots)))$$

Winding back the normal forms gives:

$$[\ T(v, j, p)\ ] \neg [\ S^{\ \forall T}(u, i, o)\ ] \neg ((G(u, v) \vee C(u, v, o, p, A)) \wedge V(u, v, o, p, A))$$

We use (10.2) on the $((G \vee C) \wedge V)$ term, and monotonicity, after which some simplification yields:

$$[\ T(v, j, p)\ ] \neg [\ S^{\ \forall T}(u, i, o)\ ] \neg (G(u, v) \wedge V(u, v, o, p, A))$$

Now it remains to existentially quantify[9] the A , and to apply the deduction principle, to arrive at (10.7). We are done. ☺

Theorem 10.6 Let (3.1) describe a ¬C retrenchment where the set of abstract operation names is Ops^M . Then (10.3), with the occurrences of $M^{\ \forall T}$ and $S^{\ \forall T}$ replaced by $M^{\ T}$ and $S^{\ T}$ respectively, where $S^{\ T}$ is the generalised substitution:

$$S^{\ T}(\ u\ ,\ i\ ,\ o\ )\ \equiv\ \text{PRE}\ \mathsf{trm}(\ T\ (\ v\ ,\ j\ ,\ p\ )\ )\ \text{THEN}\ S\ (\ u\ ,\ i\ ,\ o\ )\ \text{END} \qquad (10.8)$$

and (10.5) and (10.6), describe a normal modulated refinement.

The proof of this is a small simplification of the preceding result, insofar as in the refinement operation PO, $\mathsf{trm}(T)$ may be hypothesised directly, rather than needing a derivation step to prove it.

Theorem 10.7 Let (3.1) describe a ¬C retrenchment where the set of abstract operation names is Ops^M . Then (10.3), with the occurrences of $M^{\ \forall T}$ and $S^{\ \forall T}$ replaced by M and S respectively, and (10.5) and (10.6), describe an inverted modulated refinement.

Again the proof is a simplification of the preceding since there is no work to do for the termination part of the refinement operation PO.

The preceding three results provoke some comments about the frame issues[10] surrounding machines M , $M^{\ \forall T}$ and $M^{\ T}$. Our starting point is M , the machine given originally, whose variable is u , and whose properties involve only u (and the I/O variables i , o). It is uncontroversial to identify the M of (3.1) with the M of Theorem 10.7 since they

9. Here we used the monotonicity law $(\exists A \bullet [T(v)] \neg [S(u)] \neg F(A)) \ \Rightarrow\ [T(v)] \neg [S(u)] \neg (\exists A \bullet F(A))$.

10. The business of frames in B is not a completely cut and dried one, see [Potet and Rouzaud (1998)]. Nevertheless the kind of problems raised there tend not to infect realistic situations.

are textually the same thing. However as we consider $M^{\forall T}$ and M^{T} , we have to ask to what extent it is reasonable to regard these as versions of M , as we would wish to do to justify viewing Theorems 10.5 and 10.6 as saying something about the original ¬C retrenchment (3.1). Both $M^{\forall T}$ and M^{T} involve the concrete variable v (and the concrete I/O).

Before discussing $M^{\forall T}$ and M^{T} let us consider first the machine N of (7.7). This machine is manufactured from the B refinement, and is ostensibly only concerned with modifying v , but nevertheless it needs to refer to u in order to do so, albeit that the reference is through a quantification. The quantification can be seen as hiding the scoped occurrences of u , indeed alpha conversion decrees that these occurrences are of an arbitrary "dummy variable", but this is slightly illusory. While u indeed occurs quantified in $\mathsf{trm}(T^{\exists S})$, other components of the quantified clause do not, namely I , J , and non-variable parts of $\mathsf{trm}(S)$. These ensure that although the identity of the quantified variable in $\mathsf{trm}(T^{\exists S})$ doesn't much matter, its meaning is still fixed, it will still have values in U . The point of all this is that despite the quantification, we must still regard u as being pertinent to N , or more accurately that variables in general taking values in U , I , O must be regarded as being pertinent to N .

The same argument now applies to $M^{\forall T}$. This time it is the concrete variable v that is quantified, but the free occurrences of I , G , J and $\mathsf{trm}(S)$, ensure that variables taking values in V , J , P must be regarded as being pertinent to $M^{\forall T}$, despite the quantification.

By contrast, in M^{T} , v appears free. Superficially therefore M^{T} looks like more of a disturbance to M than $M^{\forall T}$ is, since it is clear that variables taking values in V , J , P must be regarded outright as being pertinent to M^{T} . However the preceding discussion shows that this distinction is rather misleading. Moreover the refinement relationship between M^{T} and $N^{\exists A}$ corresponds more closely to the retrenchment relationship between M and N because the refinement PO holds for exactly those (u, i, o) , (v, j, p) pairs that verify the retrenchment PO, whereas in $M^{\forall T}$, values of (u, i, o) such that the retrenchment relationship holds for some but not all (v, j, p) are not included in the refinement relationship, as a result of the quantification. Our conclusion is thus that one cannot expect to retrieve a refinement from a ¬C retrenchment unless one is prepared to enlarge what one regards as the frame of the abstract machine, or to give up the normal orientation of the refinement.

11 ∃GVP Retrenchment and Fullness

Though an easy criterion to deal with, ¬C retrenchment is rather restrictive in that the RETRIEVES and NEVERTHELESS clauses need to completely deny the CONCEDES clause everywhere they are both true. Although this sometimes happens as in the example of Section 4 (exercise for the reader), it is by no means always appropriate, as in the example of Section 3 in which the retrieve relation is violated in the $| xx | = 10$ and $new \notin xx$ situation. Indeed whenever there is some point at which the CONCEDES and NEVERTHELESS clauses are simultaneously valid, the ¬C criterion is useless. This restrictiveness of ¬C retrenchment is due to the implicit outermost universal quantification in (10.2). To this end we explore a more delicate criterion for retrenchment, the ∃GVP criterion, in which it is not everywhere true that the properties jointly established as the result of a step at both abstract and concrete levels, annul the possibility of some further step at each level, still jointly within the tenets of retrenchment. This turns out to be equivalent to a local denial of the CONCEDES clause as Lemma 11.2 shows. Given the local nature of the assumptions, we cannot expect an arbitrary concrete execution sequence to be simulable, simply because there is no requirement for it to visit solely those concrete states in which the assumptions hold, given that they do not hold everywhere. We must therefore restrict our attention to those concrete execution sequences that do so, the so-called tame ones. Even this is not yet enough to prove a stepwise simulation result, since even when the concrete execution sequence satisfies suitable properties, there is no guarantee that the abstract execution sequence will be able to cooperate. To remedy this we need to demand a condition, fullness, of the abstract system to ensure that the inductive step of the stepwise simulation theorem will go through. With this preamble, we present the technical details.

Definition 11.1 For a retrenchment like (3.1), suppose we have the stronger joint initialisation PO:

$$\begin{aligned}&[\,Y(v)\,]\neg\,[\,X(u)\,]\neg\\&\qquad(G(u,v)\wedge\bigvee_{m\,\in\,\mathsf{Ops}^M}(\forall\,j_m\;\exists\,i_m,A_m\bullet P_m(i_m,j_m,u,v,A_m)))\end{aligned}\qquad(11.1)$$

and suppose that each Ops^M operation $n \equiv (T_n, A_n, P_n, C_n, V_n)$ of N satisfies the operation compatibility PO:

$$\begin{aligned}&\neg\Big(\forall\,u,v,o,p,B\bullet((G(u,v)\vee C_n(u,v,o,p,B))\wedge V_n(u,v,o,p,B))\\&\qquad\Rightarrow\\&\qquad\neg\,(G(u,v)\wedge\bigvee_{m\,\in\,\mathsf{Ops}^M}(\forall\,j_m\;\exists\,i_m,A_m\bullet P_m(i_m,j_m,u,v,A_m)))\Big)\end{aligned}\qquad(11.2)$$

then we call the retrenchment an $\exists$GVP retrenchment.

Note that (11.2) is much more liberal than (10.2). Predicate calculus immediately yields:

Lemma 11.2 Property (10.2) holds iff (10.3) is true:

$$\begin{aligned}&(\exists\,u,v,o,p,B,m\in\mathsf{Ops}^M\bullet\\&\qquad G(u,v)\wedge V_n(u,v,o,p,B)\wedge(\forall\,j_m\;\exists\,i_m,A_m\bullet P_m(i_m,j_m,u,v,A_m)))\end{aligned}\qquad(11.3)$$

We observe that in (11.3) all effects of the CONCEDES clauses have been subsumed. This is the $\exists$GVP analogue of the $\neg$C criterion which also denies the CONCEDES clause, but this time acting more locally.

Definition 11.3 Let (3.1) describe an $\exists$GVP retrenchment where the set of abstract operation names is Ops^M . Let $\mathcal{T} \equiv [\, v_0 \text{-}(j_0, m_0, p_1)\text{-}\!\!\rightarrow v_1 \text{-}(j_1, m_1, p_2)\text{-}\!\!\rightarrow v_2 \ldots\,]$ be an execution sequence of N , with an Ops^M sequence of invoked operation names $ms \equiv [\, m_0, m_1 \ldots\,]$. Then $\mathcal{T}$ is tame iff there is a sequence $As \equiv [\, A_0, A_1 \ldots\,]$ with $\mathrm{dom}(As) = \mathrm{dom}^\bullet(\mathcal{T})$, such that for j_0 , and for suitable i_0 , A_0 witnesses (11.1), and

$$(\forall\,r\in\mathrm{dom}^\bullet(As)\bullet\Gamma(A_{m_r},v_{r+1},A_{m_{r+1}}))\qquad(11.4)$$

where

$$\begin{aligned}&\Gamma(A_{m_r},v_{r+1},A_{m_{r+1}})\;\equiv\;(\exists\,u,o,i\bullet G(u,v_{r+1})\wedge\\&\qquad V_{m_r}(u,v_{r+1},o,p_{r+1},A_{m_r})\wedge P_{m_{r+1}}(i,j_{r+1},u,v_{r+1},A_{m_{r+1}}))\end{aligned}\qquad(11.5)$$

Or in plain language, we can choose As consistently per operation instance so that at each non-boundary state of $\mathcal{T}$, (11.3) is satisfied. Note that because of their roles as "memory variables", we can assume that the A_{m_r} and $A_{m_{r+1}}$ carry with them the memory of suitable u, i, j not to mention the memory of m_r and m_{r+1} . Obviously a tame execution sequence of N comes from an ∃GVP retrenchment.

The next definition describes retrenchments in which for a given concrete step, the abstract machine is actually capable of all steps that the RETRIEVES, CONCEDES and NEVERTHELESS clauses might lead you to expect it to be capable of.

Definition 11.4 Let (3.1) describe a retrenchment where the set of abstract operation names is Ops^M . We say that M is full (with respect to the retrenchment) iff the following holds for all $m \in \mathsf{Ops}^M$:

$$\begin{aligned} & G(u, v) \wedge P(i, j, u, v, A) \wedge \mathsf{stp}(T)(v, j, v', p) \wedge \\ & \qquad (G(u', v') \vee C(u', v', o, p, A)) \wedge V(u', v', o, p, A) \\ & \quad \Rightarrow \\ & \mathsf{stp}(S)(u, i, u', o) \end{aligned} \tag{11.6}$$

The fullness criterion is not a very demanding one in actual fact. Given an arbitrary retrenchment (3.1), one can construct a corresponding retrenchment with M full, by adding fresh logical variables Au , Ai to the LVAR clause, conjoining fresh definitions

$$Au = u \wedge Ai = i$$

to the WITHIN clause, and conjoining (the body of the definition of)

$$\mathsf{stp}(S)(Au, Ai, u, o)$$

to the NEVERTHELESS clause of each operation m . This yields a retrenchment with M full because:

(a) every abstract step (including any whose existence is asserted via the retrenchment operation PO) satisfies $\mathsf{stp}(S)$, therefore strengthening the consequent of the operation PO with something that is already true cannot cause it to fail, and thus we have a valid retrenchment;

(b) the construction suggested is tantamount to conjoining the consequent of (11.6) to the antecedents, making it vacuuously true, and thus we have a retrenchment with M full.

Now we can prove the main simulation result. Note that this is related to the U-simulation discussed in [de Roever and Engelhardt (1998)].

Theorem 11.5 Let (3.1) describe an ∃GVP retrenchment where the set of abstract operation names is Ops^M . Suppose that M is full. Let $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-}\rangle\ v_1 \text{ -}(j_1, m_1, p_2)\text{-}\rangle\ v_2 \dots]$ be a tame trm-safe execution sequence of N . Then there is a stepwise simulation $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-}\rangle\ u_1 \text{ -}(i_1, m_1, o_2)\text{-}\rangle\ u_2 \dots]$ of $\mathcal{T}$.

Proof. This follows the pattern of our previous results so we just focus on the inductive step. Suppose $\mathcal{S}$ has been constructed as far as the r'th step and that we have established $G(u_r, v_r) \wedge P_{m_r}(u_r, v_r, i_r, j_r, A_r)$. Because $\mathcal{T}$ is tame, for the same choice of A_r , we have that $v_r \text{ -}(j_r, m_r, p_{r+1})\text{-}\rangle\ v_{r+1}$ witnesses (11.2)-(11.3) for appropriate u_{r+1}, i_{r+1}, o_{r+1} , A_{r+1} . But now we have the antecedents of (11.6), whose consequent yields a step $u_r \text{ -}(i_r, m_r, o_{r+1})\text{-}\rangle\ u_{r+1}$ that both verifies the stepwise simulation property, and also reestablishes the inductive hypothesis $G(u_{r+1}, v_{r+1}) \wedge P_{m_{r+1}}(u_{r+1}, v_{r+1}, i_{r+1}, j_{r+1}, A_{r+1})$. We merely add now that tameness enables us to take note of (11.1) for the initialisation, and that we can use trm-safety and the normal retrenchment operation PO for the last step (if T indeed has a last step). ☺

∃GVP retrenchment gives us a notion of strong simulation, though not of the same ilk as we have had previously. The main thing we lose is the equality of automata theoretic an machine based state that we emphasised before. We comment further after the technical details.

Definition 11.6 Let M and N be machines in an ∃GVP retrenchment with operation name sets Ops^M and Ops^N . Then taking $*$ as a fresh constant, the trm-safe tame M-reachable transition system of N is defined by:

(1) $\boldsymbol{L}_{N_M} = \{(j, m, p) \in \mathsf{J} \times \mathsf{Ops}^M \times \mathsf{P}\}$
(2) $\boldsymbol{V}^t_M = \boldsymbol{V}^t_{M0} \cup \{(A_n, v, A_m) \mid$ there is a trm-safe tame execution sequence of N consisting solely of Ops^M operations, with v as penultimate state and such that $\Gamma(A_n, v, A_m)$ holds$\}$
(3) $\boldsymbol{V}^t_{M0} = \{(*, v_0, A_{m_0}) \mid \exists\, u_0 \bullet (11.1)$ holds$\}$
(4) $T^t_{N_M} = \{(A_n, v, A_m)$ -(j, m, p)-› $(A_m, v', A_{n'}) \mid$ there is a trm-safe tame execution sequence of N consisting solely of Ops^M operations, with v -(j, m, p)-› v' as penultimate step$\}$

Theorem 11.7 Let (3.1) describe an ∃GVP retrenchment where the set of abstract operation names is Ops^M and where M is full. Then there is a strong simulation from the trm-safe tame M-reachable transition system $T^t_{N_M}$ of N to the trm-safe reachable transition system T_M of M .

Proof. We define a strong simulation $(\Theta_\mathrm{S}, \Theta_\mathrm{L})$ where Θ_L is unchanged from Theorem 10.4 so we do not quote it.

$$\Theta_\mathrm{S} = \{(u, (A_n, v, A_m)) \in \boldsymbol{U} \times \boldsymbol{V}^t_M \mid G(u, v) \wedge P_m(i_m, j_m, u, v, A_m)\}$$

Note that this is in fact well defined as A_m subsumes i_m, j_m as assumed previously. Proving that this gives a strong simulation is now a straightforward adaptation of the proof of Theorem 11.5. ☺

As is clear, the automata theoretic states in the above construction are no longer the same states as the machine states. One could imagine trying to repair this by going from the transition system $T^t_{N_M}$ to one where the states are purely elements $v \in \mathsf{V}$ as before, by existentially quantifying over the A_n and A_m in the triples (A_n, v, A_m) . However while this might well enable us to prove the strong simulation properties of $(\Theta_\mathrm{S}, \Theta_\mathrm{L})$, we would lose an essential automata theoretic property hitherto taken for granted, namely that an arbitrary sequential composition of steps through a series of states of the automaton is a valid execution sequence fragment — bluntly, an arbitrary sequential composition of steps of an automaton as suggested is not guaranteed to be tame. For tameness we need the extra items A_n and A_m in the states, and both are needed to ensure that we can relate successive steps correctly in an execution sequence. In the terminology of [Abadi and Lamport (1991)], in a typical state (A_n, v, A_m) , A_n is a history variable and A_m is a prophecy variable. Unlike most applications of history and prophecy variables, both are only of limited reach here, i.e. we need to incorporate neither the full history of the execution thus far, nor the whole of its evolution in the future, to achieve our objectives, because of the relatively local nature of the tameness criterion.

The fact that in (11.3) all effects of the CONCEDES clauses are subsumed, gives rise to the hope that some notion of modulated refinement could be recovered from an ∃GVP retrenchment. In fact the existential nature of the ∃GVP criterion acts against this as the absence of any reference to sets of execution sequences makes refinement notions intrinsically universal in nature. The next section sets out fresh criteria for retrenchment that are intended to alleviate this.

12 Decomposed Retrenchment and Trimness

In this section we make a different set of assumptions about retrenchments, and show that they can give rise to a modulated refinement property. In order that this goes through, we know that the assumptions must be essentially universal in character, from remarks in the last section. We will assume that the WITHIN clauses are decomposed, i.e. fall into two pieces, one essentially to do with I/O remapping, and the remainder; and that the I/O remapping part does not conflict with the RETRIEVE clause (see (12.2)). We also assume that the CONCEDES clauses do not overlap with the RETRIEVE clause in a particular way (12.3), a property we call trimness. We argue that both properties might be expected to arise naturally in realistic systems — this is really just separation of concerns at a human level. The natural tendency would be for designers to write down the various different properties they require individually, leading to easy identification of the various subclauses described, rather than forcing a mixture by writing clauses that deliberately mix different issues.

Definition 12.1 Let (3.1) describe a retrenchment where the set of abstract operation names is Ops^M . A WITHIN-decomposition of (3.1) consists of a collection of predicates $\{P^{\forall}_m , P^{+}_m \mid m \in \mathsf{Ops}^M\}$ such that

$$P_m(i_m, j_m, u, v, A_m) \;\Leftrightarrow\; P^{\forall}_m (i_m, j_m, u, v) \wedge P^{+}_m (i_m, j_m, u, v, A_m) \tag{12.1}$$

and

$$G(u, v) \Rightarrow \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m \,\exists\, i_m \bullet P^{\forall}_m (i_m, j_m, u, v)) \tag{12.2}$$

We refer to a retrenchment as a decomposed retrenchment if we have some such decomposition in mind.

Every retrenchment has a trivial WITHIN-decomposition since we can always set $P^{\forall}_m \equiv \mathit{true}$ and $P^{+}_m \equiv P_m$, however this is obviously not going to be terribly useful. What we have in mind rather, is when a retrenchment step defines a remapping of I/O and state aspects of a system, and also some further stipulations on the permitted starting configurations of operations. In a sane design process these two aspects will be described separately, so in a given P_m , there will be clauses pertaining to the former activity and others pertaining to the latter. These can be gathered into $P^{\forall}_m$ and P^{+}_m respectively, so $P_m \equiv P^{\forall}_m \wedge P^{+}_m$. In particular, the global nature of a sensible remapping of I/O and state would lead to the global applicability of $P^{\forall}_m$ and hence to (12.2).

Definition 12.2 Let (3.1) describe a retrenchment where the set of abstract operation names is Ops^M . We say the retrenchment is trim iff for all $m \in \mathsf{Ops}^M$:

$$(\exists\, u \bullet G(u, v)) \Rightarrow \neg\, (\exists\, u \bullet C_m(u, v, o, p, A)) \tag{12.3}$$

Trimness just means that all CONCEDES clauses are disjoint in a particularly clean way from the RETRIEVES clause. Essentially (12.3) divides V into a part G-related to U , a part C-related to U , and any remainder.

Any retrenchment which can withstand the strengthening of the NEVERTHELESS clause by the term $G^{\wedge} \equiv (G(u, v) \vee \neg (\exists\, u \bullet G(u, v)))$ can be made trim by replacing each clause $C_{m\text{-old}} \equiv C_m(u, v, o, p, A)$ by $C_{m\text{-new}} \equiv (C_m(u, v, o, p, A) \wedge \neg (\exists\, u \bullet G(u, v)))$, since $((G(u, v) \vee C_{m\text{-new}}) \wedge V_m) \Leftrightarrow ((G(u, v) \vee C_{m\text{-old}}) \wedge V_m \wedge G^{\wedge})$, and $C_{m\text{-new}}$ validates (12.3) regardless of what $C_{m\text{-old}}$ is. From this we see that the the $C_{m\text{-old}}$ to $C_{m\text{-new}}$ transformation maintains the truth of all the necessary POs.

The main interesting property of decomposed trim retrenchments is that a useful notion of modulated refinement can be recovered from them. Since loosely speaking, any retrenchment can be decomposed at least trivially, and any retrenchment has a trim counterpart as above if it is not trim already, the relevant theorems are of wide applicability; indeed Theorems 12.3 and 12.4 are arguably the most important theorems in this paper.

Theorem 12.3 Let (3.1) describe a decomposed trim retrenchment where the set of abstract operation names is Ops^M . Then the following is a normal modulated refinement:

```
MACHINE          $M^{\exists P}(a)$                 MACHINE          $N^{\exists V}(b)$
                                        MODREF           $M^{\exists P}$
VARIABLES        $u$                      VARIABLES        $v$
INVARIANT        $I(u)$                   INVARIANT        $J(v)$
                                        RETRIEVES        $G(u, v)$
INITIALISATION   $X(u)$                   INITIALISATION   $Y(v)$
OPERATIONS                              OPERATIONS
    $o \longleftarrow OpName(i) \triangleq$            $p \longleftarrow OpName(j) \triangleq$
        $S^{\exists P}(u, i, o)$                     BEGIN
END                                             $T^{\exists V}(v, j, p)$
                                            WITHIN
                                                $P^{\forall}(i, j, u, v)$
                                            NEVERTHELESS
                                                $V^{\exists A}(u, v, o, p)$
                                            END
                                        END                              (12.4)
```

where:

```
$S^{\exists P}(u, i, o) \equiv$ PRE
            $(\forall\, v, j \bullet I(u) \wedge G(u, v) \wedge J(v) \wedge \mathrm{trm}(S(u, i, o))$
                $\Rightarrow \mathrm{trm}(T(v, j, p)) \wedge (\exists\, A \bullet P^{+}(i, j, u, v, A)))$
        THEN
            $S(u, i, o)$
        END                                                              (12.5)

$T^{\exists V}(v, j, p) \equiv$ PRE
            $\mathrm{trm}(T(v, j, p))$
        THEN
            ANY
                $v', p'$
            WHERE
                $\mathrm{stp}(T)(v, j, v', p') \wedge (\exists\, u' \bullet G(u', v'))$
            THEN
                $v, p := v', p'$
```

END
END (12.6)

$$V^{\exists A}(u, v, o, p) \equiv (\exists A \bullet V(u, v, o, p, A)) \qquad (12.7)$$

Proof. We have two fresh machines to deal with so we must check their consistency first. For $M^{\exists P}$ we observe that its invariant I and initialisation X are unchanged from M so the initialisation PO holds; and because the trm predicates of its Ops^M operations are at least as strong as those of M, and within these trm predicates the stp predicates are unchanged, if the operation POs of M can be discharged then so can those of $M^{\exists P}$. Likewise for $N^{\exists V}$, its invariant J and initialisation Y are unchanged from N; and because its trm and stp predicates are at least as strong than those of N, discharging of the operation POs of N leads to the discharge of those of $N^{\exists V}$.

Let Φ name the PRE clause $(\forall v, j \bullet \dots P^{+})$ in (12.20), and let Ψ name the WHERE clause $\mathsf{stp}(T) \wedge G$ in (12.6).

We turn to the retrenchment itself. The joint initialisation PO is unproblematic, by (12.2). Likewise the operation compatibility PO (8.4) follows since G alone implies the conjunction of all the $P^{\forall}$ predicates by (12.2) again. There remains the PO for operations. For this we need to show that for all operations in Ops^M:

$$\begin{aligned}&(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(S^{\exists P}(u, i, o)) \wedge P^{\forall}(i, j, u, v))\\ &\quad\Rightarrow\\ &\quad \mathsf{trm}(T^{\exists V}(v, j, p)) \wedge [\, T^{\exists V}(v, j, p)\,] \neg [\, S^{\exists P}(u, i, o)\,] \neg\\ &\qquad\quad (G(u, v) \wedge V^{\exists A}(u, v, o, p))\end{aligned} \qquad (12.8)$$

knowing that (3.7) and (12.1)-(12.3) hold.

First we observe that $\mathsf{trm}(T^{\exists V}) = \mathsf{trm}(T)$. This is because $\mathsf{trm}(v, p := v', p') = \mathit{true}$, $\mathsf{trm}(\text{ANY } v' \text{ WHERE } Q \text{ THEN } S \text{ END}) = (\forall v' \bullet Q \Rightarrow \mathsf{trm}(S))$, and $\mathsf{trm}(\text{PRE } Q \text{ THEN } S \text{ END}) = Q \wedge \mathsf{trm}(S)$.

Now we hypothesise the antecedents of (12.8). Using generalised modus ponens, we can infer $\mathsf{trm}(T(v, j, p))$. Since $\mathsf{trm}(T) = \mathsf{trm}(T^{\exists V})$, we have the first part of the consequent of (12.8).

For the second part, suppose we have a step of $T^{\exists V}$, $v_0 \text{ -}(j_0, T^{\exists V}, p_0)\text{-›}\, v_0'$. Then we know that $v_0 \text{ -}(j_0, T, p_0)\text{-›}\, v_0'$ is a step of T, that $P^{\forall}(i_0, j_0, u_0, v_0)$ is hypothesised, and that we can deduce $\exists A \bullet P^{+}(i, j, u, v, A)$ from the hypothesised Φ. This means that we can find an A_0 such that $P(i, j, u, v, A_0)$ holds. We now have all the antecedents of the retrenchment operation PO, (3.7). This generates a step $u_0 \text{ -}(i_0, S, o_0)\text{-›}\, u_0'$ of M which we can reinterpret as a step $u_0 \text{ -}(i_0, S^{\exists P}, o_0)\text{-›}\, u_0'$ of $M^{\exists P}$ since we are hypothesising Φ. For this step of $M^{\exists P}$, $((G(u_0', v_0') \vee C(u_0', v_0', o_0, p_0, A_0)) \wedge V(u_0', v_0', o_0, p_0, A_0))$ is true. From the structure of Ψ we know that $G(u_0^{\sim}, v_0')$ is true for some $u_0^{\sim}$, and from (12.3) we know $G(u_0^{\sim}, v_0') \Rightarrow \neg\, C(u_0', v_0', o_0, p_0, A_0)$. After some algebra and weakening we derive $G(u_0', v_0') \wedge V(u_0', v_0', o_0, p_0, A_0)$. We can now existentially quantify the A; and noting that we have both the trm predicates and the relation on steps, allows us to apply Theorem 5.1, and then quickly to get (12.8). So we are done. ☺

Theorem 12.4 Let (3.1) describe a decomposed trim retrenchment where the set of abstract operation names is Ops^M. Then (12.4), with the occurrences of $M^{\exists P}$ and $S^{\exists P}$ replaced by M^P and S^P respectively, where S^P is the generalised substitution:

$$S^{P}(u,i,o) \equiv \text{PRE}\ \mathsf{trm}(T(v,j,p)) \wedge (\exists A \bullet P^{+}(i,j,u,v,A)) \\ \text{THEN}\ S(u,i,o)\ \text{END} \tag{12.9}$$

and (12.6) and (12.7), describe a normal modulated refinement.

The proof of this is a simple matter of managing without the universal quantifiers in the PRE component. An equally simple modification suffices for the following.

Theorem 12.5 Let (3.1) describe a decomposed trim retrenchment where the set of abstract operation names is Ops^M. Then (12.4), with the occurrences of $M^{\exists P}$ and $S^{\exists P}$ replaced by M and S respectively, and (12.6) and (12.7), describe an inverted modulated refinement.

The fact that we can derive a modulated refinement from a decomposed tidy retrenchment means that all the results in Section 8 are available to us. In particular we have the stepwise and strong simulation properties. These could also have been derived independently of course, but while the strong simulation property is the same whichever way it is derived, the independently derived stepwise simulation property is different to the one inherited from modulated refinement as a comparison of (8.5) and (9.1) shows (clearly under our assumptions both are true). All this would be very similar to what appears in Section 10 which can be seen as a partial working out of these remarks for the ¬C case, so we suppress the details.

13 Other Variations

In the preceding sections we looked at a variety of types of retrenchment for which we were able to demonstrate different kinds of simulation as appropriate. The possibilities discussed do not by any means exhaust what can be proposed in this area. However, as the preceding sections showed, much of the reasoning involved is very similar; in fact in all cases it largely hinges on denying the CONCEDES clause in different ways. This is hardly surprising considering that $G \wedge P$ has to be established for a next step. In this section we consider some variations on the themes already proposed, but briefly, to avoid excessive repetition.

∃GVP retrenchment has some variations worth mentioning. In the first, v is removed from the scope of the universal quantifier in (11.2) (or equivalently from the scope of the existential quantifier in (11.3)). This has the effect of making these clauses hold for all $v \in \mathsf{V}$, which increases the proportion of behaviours of the machine N that can be simulated. Removing also j from quantification results in a further increase, and so on. A different kind of variation replaces the disjunction over $m \in \mathsf{Ops}^M$ in (11.1) and (11.2) by a conjunction. This also increases the proportion of behaviours of N that can be simulated, and the resulting theory looks much more like that of ¬C retrenchment. In particular a modulated refinement result can be proved relatively painlessly (because the definition of a modulated refinement demands such a conjunction). Yet another variation strengthens the fullness and tameness conditions by disallowing the logical constants A introduced in LVAR clauses, giving rise to a more local version of the theory. The most striking consequence of this is to reestablish the correspondence between automata theoretic and machine oriented state that we had before. Obviously these variations can be considered separately or together.

Decomposed trim retrenchment has two variations which we consider. In the first, we add V to the antecedent of (12.3), which in fact has very little effect. In the second, we avail ourselves of the opportunity to modify the WHERE clause of (12.6). The most obvious thing to try, is to remove the existential quantification over u' . If we do this then we can also remove the existential quantifications in (12.3) giving a fresh notion of tidiness. This results in a neater theory because now *any* retrenchment can be made tidy unconditionally, as $(G \vee C) \Leftrightarrow (G \vee (C \wedge \neg G))$. However the price to pay for this is having u' free in (12.6), which raises frame problems again. These appear more severe than previously since now the extraneous variable appears directly in the definition of the result of the operation rather than merely helping to control when the operation may be called. For this reason we shied away from developing this version of the theory in detail above, despite its obvious technical appeal.

14 Conclusions

In the preceding sections we introduced and briefly justified the sharp form of retrenchment, and we compared it with the simpler unsharp form. Then we proved that retrenchments compose. In fact a comparison between the composition proofs for unsharp and sharp forms leads us to conjecture that compositionality holds regardless of the shape of the formula Θ established by the $[T(v, j, p)]\neg[S(u, i, o)]\neg\ \Theta$ complex. Any Θ built up out of a collection of clauses which occur once only in the operation PO, will lead to the compositionality property by suitable manipulation, essentially because the "single occurrence in the PO" property does not constrain *how* the component clauses can be manipulated in order to generate a composed Θ . Thus one could imagine increasingly elaborate forms of retrenchment, and thus of Θ , beyond the $\Theta \equiv (G \vee C)$ or $\Theta \equiv ((G \vee C) \wedge V)$ forms examined already, and anticipate that they would compose without problems. However we can see little need at this juncture to do so.

Beyond compositionality, we examined the relationship between retrenchment and refinement in some detail. While refinement was easily encompassed within retrenchment, we found that modulated refinement provided us with the natural environment in which to explore the converse relationship. We found that in useful special cases one could find a refinement hiding within a retrenchment. To an extent one could argue that some of the derivations here amounted to just shuffling of formulae around the concrete syntax of the retrenchment and modulated refinement frameworks, an option afforded to us by an element of expressive redundancy in these syntactic frameworks. On purely mathematical grounds such an argument is hard to fault; however it glosses over an important point. In industrial scale developments, one of the main enemies of fluent specification is the sheer size of the text. By the time one has incorporated all the irksome real-world details that need to be taken into account, the specification text has become unwieldy and opaque, and difficult to approach from cold. Furthermore, familiar specification composition constructs (such as INCLUDES and USES in B, and analogues of these in other specification formalisms) turn out to be less useful than one would wish in such situations because of their extreme insistence that the theory of the subordinate construct remain unaffected by the environment into which it is placed. Mathematically this insistence is highly laudable, but in practice it often leads to developments that look upside down from an engineering viewpoint, as low level details take on a dominant role due to the pervasiveness of their consequences. Needless to say there is a certain danger when real world developments are forced into what is, from an engineering standpoint, a distorted perspective. Retrenchment attempts to redress the balance to some extent, by allowing deviation from previously described behaviour, such previously described behaviour being sacrosanct from the refinement point of view. In this regard, the manipulations of eg. Theorems 10.5 and 12.3 are evidence that retrenchment is helping to achieve a separation of concerns, whereby properties that might be more awkwardly expressed using refinement, may be distributed around a retrenchment to make the latter a means for easier understanding of the concrete specification. Moreover our reference point, modulated refinement itself, is of intrinsic interest as it allows the I/O signature of operations to change, a feature that is of some value in industrial scale developments ([Meynadier (1998)]).

Another major area of interest to us was simulation. Here the subtleties of the various notions of simulation that we considered, were shown up in the different simulation properties possessed by different kinds of retrenchment and refinement notions that we discussed. Unsurprisingly the widest range of notions of simulation was displayed by $\neg$C retrenchment, a kind of retrenchment with properties strong enough to almost make it a refinement. For this special case, we were able to display stepwise simulation, which translated smoothly into strong simulation, and we furthermore extracted three closely related types of modulated refinement. By contrast, for the fairly weakly constrained notion of $\exists$GVP retrenchment, while we could prove stepwise simulation for tame execution sequences, strong simulation appeared in a muted form due to the necessity of incorporating limited reach history and prophecy variables. Also the disjunctive nature of the assumed restrictions prevented an immediate extraction of a modulated refinement; this was more easily accomplished for decomposed retrenchments which feature suitably conjunctive assumptions. And the whole area is seen in the highest relief in modulated refinement itself, which offers a route between the simulation properties of conventional refinement and those of strong simulation. Our approach to these issues may be contrasted with the work of [Bolton et al. (1999), Derrick et al. (1996)] who consider related questions.

One area we deliberately neglected in this paper was event driven systems, a popular current topic of research in the B-Method, where they are typically refered to as Abstract Systems rather than Abstract Machines (see eg. [Abrial and Mussat (1998), Abrial (1996b), Butler and Walden (1996), Walden and Sere (1996)]). These works build on [Back and Kurki-Suonio (1983), Back (1989), Lamport (1994)], and others. The interplay between the ideas inherent in action refinement and retrenchment is certainly worth studying, but we leave that to another place. And yet another subject for future work, is elaborating the ideas of this paper in the context of other refinement formalisms.

References

Abadi M., Lamport L. (1991); The Existence of Refinement Mappings. Theor. Comp. Sci. **82**, 253-284.

Abrial J. R. (1996a); The B-Book. Cambridge University Press.

Abrial J. R. (1996b); Extending B without Changing it: (for Developing Distributed Systems). *in*: Proc. B-96, Habrias (ed.), 169-190, IRIN Nantes, France, ISBN 2-906082-25-2.

Abrial J. R., Mussat L. (1998); Introducing Dynamic Constraints in B. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 83-128, Springer.

Back R. J. R. (1989); Refinement Calculus Part II: Parallel and Reactive Systems. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), LNCS **430**, 67-93, Springer.

Back R. J. R., Kurki-Suonio R. (1983); Decentralisation of Process Nets with Centralised Control. *in*: Proc. 2nd ACM SIGACT-SIGOPS Symp. on Princ. Dist. Comp., 131-142, ACM.

Banach R., Poppleton M. (1998); Retrenchment: An Engineering Variation on Refinement. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 129-147, Springer. *See also*: UMCS Tech. Rep. UMCS-99-3-2, `http://www.cs.man.ac.uk/cstechrep`

Behm P., Burdy L., Meynadier J.-M. (1998); Well Defined B. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 29-45, Springer.

Blikle A. (1981); The Clean Termination of Iterative Programs. Acta Inf. **16**, 199-217.

Boiten E., Derrick J. (1998); IO-Refinement in Z. *in*: Proc. Third BCS-FACS Northern Formal Methods Workshop. Ilkley, U.K., B.C.S. `http://www.ewic.org.uk/ewic/workshop/view.cfm/NFM-98`

Butler M., Walden M. (1996); Distributed Systems Development in B. *in*: Proc. B-96, Habrias (ed.), 155-168, IRIN Nantes, France, ISBN 2-906082-25-2.

Bolton C., Davies J., Woodcock J. (1999); On the Refinement and Simulation of Data Types and Processes. *in*: Proc. IFM-99, Araki, Galloway, Taguchi (eds.), 273-292, Springer.

Coleman D., Hughes J. W. (1979); The Clean Termination of Pascal Programs. Acta Inf. **11**, 195-210.

Derrick J., Bowman H., Boiten E., Steen M. (1996); Comparing LOTOS and Z Refinement Relations. *in*: Proc. FORTE/PSTV-9, 501-516, Chapman and Hall.

de Roever W-P., Engelhardt K. (1998); Data Refinement: Model-Oriented Proof Methods and their Comparison. Cambridge University Press.

Dunne S., Galloway A., Stoddart B. (1998); Specification and Refinement in General Correctness. *in*: Proc. Third BCS-FACS Northern Formal Methods Workshop. Ilkley, U.K., B.C.S. `http://www.ewic.org.uk/ewic/workshop/view.cfm/NFM-98`

Hayes I. J. (1993); Specification Case Studies. (2nd ed.), Prentice-Hall.

Hayes I. J., Sanders J. W. (1995); Specification by Interface Separation. Form. Asp. Comp. **7**, 430-439.

Jones C. B. (1983); Tentative Steps Towards a Development Method for Interfering Programs. ACM Trans. Prog. Lang. Sys. **5**, 596-619.

Jones C. B. (1990); Systematic Software Development Using VDM. (2nd ed.), Prentice-Hall.

Jones C. B., Shaw R. C. (1990); Case Studies in Systematic Software Development. Prentice-Hall.

Jonsson B. (1989); On Decomposing and Refining Specifications of Distributed Systems. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), LNCS **430**, 361-385, Springer.

Jonsson B. (1991); Simulations between Specifications of Distributed Systems. *in*: Proc. CONCUR-91, Baeten, Groote (eds.), LNCS **527**, 346-360, Springer.

Lamport L. (1994); A Temporal Logic of Actions. ACM Trans. Prog. Lang. Sys. **16**, 872-923.

Lano K., Haughton H. (1996); Specification in B: An Introduction Using the B-Toolkit. Imperial College Press.

Lynch N. (1989); Multivalued Possibilities Mappings. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), LNCS **430**, 519-543, Springer.

Merritt M. (1989); Completeness Theorems for Automata. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), LNCS **430**, 544-560, Springer.

Mery D. (1996); Machines Abstraites Temporelles, Analyse Comparative de B et TLA$^+$. *in*: Proc. B-96, Habrias (ed.), 191-220, IRIN Nantes, France, ISBN 2-906082-25-2.

Meynadier J.-M. (1998); Private Communication.

Mikhajlova A, Sekerinski E. (1997); Class Refinement and Interface Refinement in Object-Oriented Programs. *in*: Proc. FME-97, Fitzgerald, Jones, Lucas (eds.), LNCS **1313**, 82-101, Springer.

Neilson D. S. (1990); From Z to C: Illustration of a Rigorous Development Method. PhD. Thesis, Oxford University Computing Laboratory Programming Research Group, Technical Monograph PRG-101.

Owe O. (1985); An Approach to Program Reasoning Based on a First Order Logic for Partial Functions. University of Oslo Institute of Informatics Research Report No. 89. ISBN 82-90230-88-5.

Owe O. (1993); Partial Logics Reconsidered: A Conservative Approach. Form. Asp. Comp. **3**, 1-16.

Potet M-L., Rouzaud Y. (1998); Composition and Refinement in the B-Method. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 46-65, Springer.

Sekerinski E., Sere K. (1998); Program Development by Refinement: Case Studies Using the B Method. Springer.

Spivey J. M. (1993); The Z Notation: A Reference Manual. (2nd ed.), Prentice-Hall.

Stepney S., Cooper D., Woodcock J. (1998); More Powerful Z Data Refinement: Pushing the State of the Art in Industrial Refinement. *in*: Proc. ZUM-98, Bowen, Fett, Hinchey (eds.), LNCS **1493**, 284-307, Springer.

Walden M., Sere K. (1996); Refining Action Systems within B-Tool. *in*: Proc. FME-96, M.-C. Gaudel, J. Woodcock (eds.), LNCS **1051**, 85-104, Springer.

Woodcock J., Davies J. (1996); Using Z: Specification, Refinement, and Proof. Prentice-Hall.

Wordsworth J. B. (1996); Software Engineering with B. Addison-Wesley.