Retrenchment, 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 Comp., Open University, Milton Keynes, MK7 6AL, U.K.
banach@cs.man.ac.uk , m.r.poppleton@open.ac.uk

Abstract: Retrenchment is introduced as a liberalisation of refinement intended to address some of the shortcomings of refinement as sole means of progressing from simple abstract models to more complex and realistic ones. In retrenchment the relationship between an abstract operation and its concrete counterpart is mediated by extra predicates, allowing the expression of non-refinement-like properties and the mixing of I/O and state aspects in the passage between levels of abstraction. Modulated refinement is introduced as a version of refinement allowing mixing of I/O and state aspects, in order to facilitate comparison between retrenchment and refinement, and various notions of simulation are considered in this context. Stepwise simulation, the ability of the simulator to mimic a sequence of execution steps of the simulatee in a sequence of equal length is proposed as the benchmark semantic notion for relating concepts in this area. One version of modulated refinement is shown to have particularly strong connections with automata theoretic strong simulation, in which states and step labels are mapped independently from simulator to simulatee. A special case of retrenchment, simple simulable retrenchment is introduced, and shown to have properties very close to those of modulated refinement. The more general situation is discussed briefly. 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 [1] the authors observed that the normal practice of using refinement as the sole means of going from an abstract description of a desired system to a more realistic one, exhibited certain deficiencies as regards the desirability of keeping things simple and elegant at the highest levels of description, whilst accepting that a lower level account needs to recognise the impact of many low level details that necessarily intrude, in an essential way, upon the idealised nature of the former. We therefore proposed that the exigencies of refinement were mollified by two extra predicates per operation, the WITHIN and CONCEDES clauses, the former to strengthen the precondition and the latter to weaken the postcondition, the latter in particular allowing the expression of non-refinement-like behaviour because of the weakening of the postcondition. Permitting these clauses to also mix state and I/O information between levels of abstraction when convenient, yields a very flexible framework for building up complex specifications from over-simple but appealing predecessors. In this manner we overcame the unforgiving nature of the refinement proof obligations.

In [1] we were concerned with justifying retrenchment on engineering grounds. This more pragmatic departure we considered reasonable, so that we did not fall into the trap of making a premature commitment to a particular mathematical notion that later proved to be inconvenient in the face of large examples. In the present work we return to examine the foundations of the notion that we have proposed. Specifically we examine stepwise simulation, the ability to simulate a sequence of steps of the simulatee by an equal length sequence of steps of the simulator. The main tool for this is a notion of refinement, called modulated refinement, similar to elaborations of conventional refinement that allow for change of I/O representations. Modulated refinement comes in two versions, normal and inverted, and the latter supports an especially strong connection with (automata theoretic) strong simulation. This is of independent interest, and furthermore provides the means to show how the properties of retrenchment are related to those of refinement.

The rest of this paper is as follows. In Section 2 we discuss via an example how refinement can be inconvenient in developing complex specifications from simpler models. We also discuss some ways in which the existing literature addresses these points, if, in our view, only partially. In Section 3 we show how retrenchment provides a natural framework for the needed flexibility. Section 4 highlights the point that stepwise simulation is the fundamental semantic notion by which we measure the relationships between systems considered in this paper. Section 5 introduces modulated refinement in its two versions, and elaborates the connection between these and (automata theoretic) strong simulation. The link between refinement and retrenchment is considered in Section 6, which introduces simple simulable retrenchment, a special case having properties very close to those of modulated refinement. Specifically, stepwise simulation and strong simulation results are easy to derive, and modulated refinements of the two kinds are recovered. Section 7 returns to the original example and Section 8 concludes.

Notation. In the body of the paper we use the B Abstract Machine Notation for model oriented specification and system development (see [2, 3, 4, 5]). This provides a comprehensive syntax and semantics for the concepts of refinement most used in development, and our ideas slot very neatly into the B framework. Nevertheless the ideas of the paper are independent of notation, and readily apply to other approaches.

2 Some Inadequacies of Refinement

While refinement has proved its worth many times over as an implementation mechanism, there is room for misgivings in its use to describe the much more informal processes that often occur when moving from an appealing and simple model of a system, to a realistic but more complex and less elegant one, where it is the latter that must actually be implemented. Let us illustrate with a small example. We will consider a mobile radio system. At a high level of abstraction we can model its essential features thus:

```
MACHINE          Mobile_Radio_HL
SETS             CALLSTATES = { Idle , Busy }
VARIABLES        callState , currChan
INVARIANT        callState ∈ CALLSTATES ∧ currChan ∈ CHANELS
INITIALISATION   callState := Idle || currChan :∈ CHANELS
OPERATIONS
    call_outgoing ( num ) ≙
        PRE  callState = Idle ∧ num ∈ CHANELS
        THEN
            CHOICE  callState := Busy || currChan := num
            OR  skip
```

```
            END
        END ;
    call_incoming ( num ) ≙
        PRE  num ∈ CHANELS
        THEN
            SELECT  callState = Idle
                THEN  callState := Busy || currChan := num
            ELSE  skip
            END
        END ;
    disconnect_outgoing ≙
        PRE  callState = Busy  THEN  callState := Idle  END ;
    disconnect_incoming ≙
        SELECT  callState = Busy  THEN  callState := Idle  END ;
END
```

The model describes what we would expect 'normal service' to consist of. In this model we distinguish between *outgoing* operations that are initiated by the current possessor of the device in question and whose validity is protected by PRE assertions, and *incoming* operations that are prompted unpredictably from the (user's) environment and whose validity (besides input typing clauses) is protected by SELECT guards. In the former case the operation diverges if called outside its PRE clause, in the latter case it will not start unless the SELECT guard is true. The distinction is made clear by considering the normal form of an arbitrary AMN operation as described in [2] Ch. 6, which can be written as:

```
opname ≙
    PRE  P(x)
    THEN
        ANY  x′  WHERE  Q(x, x′)  THEN  x := x′  END
    END                                                    (2.1)
```

for suitable predicates P and Q in the variables mentioned. For *opname* to guarantee to establish some property Π , written [*opname*] Π , the following must hold:

$$P(x) \wedge (\forall x' \bullet \neg Q(x, x') \vee \Pi[x' \backslash x]) \tag{2.2}$$

Divergence or abortion or nontermination, is caused by the failure of $P(x)$ to hold, which prevents the predicate from ever being true. Normal working is when $P(x)$ holds and there is an x' such that $Q(x, x')$ and $\Pi[x' \backslash x]$ hold. However when $P(x)$ holds and there is no x' such that $Q(x, x')$ holds, the operation succeeds miraculously since the above predicate is true independently of $\Pi[x' \backslash x]$. Since miracles are infeasable, the miraculous region of an operation is interpreted as one in which the operation cannot start, dually to the interpretation of the nonterminating region as one in which the operation cannot stop (normally). The SELECT guards $G(x)$ say, used above, just correspond to cases where $Q(x, x')$ can be decomposed into the form $G(x) \wedge x' = value$, where *value* is independent of x and x' .

Moving now to a lower level of abstraction, we take into account various facts: firstly that before the radio will work, the user must select a suitable waveband; secondly that when making an outgoing call the radio may jam, from which it must be reset; thirdly that during a call, fadeouts can occur which will also cause the radio to jam, requiring a reset. The lower level model is then as follows:

```
MACHINE           Mobile_Radio_LL
SETS              JCALLSTATES = CALLSTATES ∪ { Jam }
VARIABLES         jcallState , jcurrChan , bandSelected
INVARIANT         jcallState ∈ JCALLSTATES ∧
                      jcurrChan ∈ CHANELS ∧ bandSelected ∈ BOOL
INITIALISATION    jcallState := Idle || jcurrChan :∈ CHANELS ||
                      bandSelected := FALSE
OPERATIONS
    select_band ≙
        PRE bandSelected = FALSE THEN bandSelected := TRUE END ;
    call_outgoing ( num ) ≙
        PRE bandSelected = TRUE ∧ jcallState = Idle ∧ num ∈ CHANELS
        THEN
            CHOICE jcallState := Busy || jcurrChan := num
            OR skip
            OR jcallState := Jam
            END
        END ;
    call_incoming ( num ) ≙
        PRE num ∈ CHANELS
        THEN
            SELECT bandSelected = TRUE ∧ jcallState = Idle
                THEN jcallState := Busy || jcurrChan := num
            ELSE skip
            END
        END ;
    disconnect_outgoing ≙
        PRE bandSelected = TRUE ∧ jcallState = Busy
        THEN jcallState := Idle
        END ;
    disconnect_incoming ≙
        SELECT bandSelected = TRUE ∧ jcallState = Busy
        THEN jcallState := Idle
        END ;
    fadeout ≙
        SELECT bandSelected = TRUE ∧ jcallState = Busy
        THEN jcallState := Jam
        END ;
    reset ≙
        PRE jcallState = Jam THEN jcallState := Idle END ;
END
```

One might say very loosely that one had refined the HL model to the LL model, but one could not attach any mathematical weight to such a statement. To see this it suffices to examine the refinement proof obligation in B notation:

$$\begin{aligned} & INV_{\mathrm{A}} \wedge INV_{\mathrm{C}} \wedge \mathsf{trm}(opname_{\mathrm{A}}) \\ & \qquad \Rightarrow \mathsf{trm}(opname_{\mathrm{C}}) \wedge [opname_{\mathrm{C}}] \neg [opname_{\mathrm{A}}] \neg INV_{\mathrm{C}} \end{aligned} \tag{2.3}$$

In this the A and C subscripts indicate the more abstract and the more concrete of the models respectively, the *INV* clauses refer to the invariants at the two levels, and the

trm clauses describe the termination conditions for the operation in question (given in the PRE clauses). The heart of the refinement PO is the '$[opname_{\mathrm{C}}] \neg [opname_{\mathrm{A}}] \neg INV_{\mathrm{C}}$' clause which states that whenever the concrete operation is able to make a step, there is a step that the abstract operation is able to make such that the concrete invariant is reestablished. Now normally in B, the concrete invariant contains clauses that relate the abstract and concrete state variables, i.e. the retrieve relation. In our example, with malice aforethought, we omitted to do this, but we can easily repair the situation by explicitly defining the retrieve relation thus:

$$\text{RETRIEVES} \qquad jcallState = callState \wedge jcurrChan = currChan \tag{2.4}$$

and rewriting the PO thus:

$$\begin{aligned} & INV_{\mathrm{A}} \wedge RET_{\mathrm{AC}} \wedge INV_{\mathrm{C}} \wedge \mathsf{trm}(opname_{\mathrm{A}}) \\ & \qquad \Rightarrow \mathsf{trm}(opname_{\mathrm{C}}) \wedge [opname_{\mathrm{C}}] \neg [opname_{\mathrm{A}}] \neg RET_{\mathrm{AC}} \end{aligned} \tag{2.5}$$

We can now examine the implications of this for various operations in our example. For the moment we will disregard the fact that the more concrete model features more operations than the abstract one. Consider the operation *disconnect_outgoing* . Its concrete trm predicate ($bandSelected = \mathit{TRUE} \wedge jcallState = Busy$) is stronger than its abstract one ($jcallState = Busy$) , so the latter does not imply the former as required in the PO. One way round this is to notice that for the corresponding *incoming* operation, the trm predicates are both true , resolving one problem, and to notice that the '$[opname_{\mathrm{C}}] \neg [opname_{\mathrm{A}}] \neg RET_{\mathrm{AC}}$' structure demands that the concrete SELECT guard implies the abstract one (as can be derived from (2.2) and the remarks which follow it). Since the *incoming* operation's guards are (essentially) just the *outgoing* operation's preconditions, this succeeds. So we can model the situation desired by keeping to the *incoming* style, retaining a refinement, but we lose the distinction between the two kinds of operation.

Consider now the operation *call_outgoing* . The strengthened precondition problem is just as evident here, but beyond that, in the concrete model, if the call fails to connect, the apparatus ends up in the *Jam* state, outside the reach of the retrieve relation. Changing preconditions to guards will not help here. No notion of refinement can cope with such a situation. A similar but more behavioural manifestation of the same phenomenon is apparent in the *fadeout* operation: if a communication is in progress and a *fadeout* event occurs, there is no way that a concrete execution sequence including this can be modelled by an abstract execution sequence, again because the concrete state ends up outside the reach of the retrieve relation. This would still be the case if we introduced a dummy abstract *fadeout* operation, specified by skip , to be 'refined by' the concrete one. From the point of view of the abstract model, such an operation would be adding uninformative clutter, and more than anything would be signalling that the relationship between the 'real' abstract model and the more concrete one is certainly *not* a refinement. A case of 'skip considered harmful'.

We can go further. Given the greater range of possible behaviours of the concrete *call_outgoing* operation compared to the abstract version, we would expect the user to be given more feedback, say in the form of some output. This would require a change in operation signature viz.

$$res \longleftarrow call\_outgoing\ (\ num\ )\ \triangleq\ \ldots$$

Changes of signature are not allowed in conventional refinement. And even if we enhanced the abstract operation with output to provide user feedback, a different set of output messages would be appropriate in the two cases, again forbidden in conventional refinement.

All of the foregoing is not to say that more generous interpretations of the concept of refinement have not allowed some of the phenomena mentioned above, one way or another. For example, change of I/O representation within refinement has been admitted in [6, 7, 8, 9], though this does not extend to mixing of I/O and state concerns as we have proposed. Using *skip* to bypass awkward points of refinement where a more informative account of things is impossible, is of course a familiar trick. Likewise the device of introducing new operations at the concrete level which refine *skip* at the abstract level, is familiar from action refinement [10, 11, 12] — the actual presence or absence of dummy operations at the abstract level becomes then a matter of mere notation. Such operations also play a part in superposition refinement [13, 14, 15], where additional concrete computations are introduced to control the progress of a more abstract computation. Our introduction of the *bandSelected* variable and its influence on the abstract computation can be seen as being in the flavour of a superposition, though it cannot be a superposition in any strict sense since the abstract computation is interfered with at the more concrete level by the consequences of jamming. The jamming phenomenon itself and the way it relates to 'normal system behaviour', bears comparison with work on the difficulties of refining abstract variables which take values in ideal and typically infinite domains, to concrete variables taking values in strictly finite ones. From a wide range of approaches to the latter question we can mention [16, 17, 18, 19, 20].

Despite these efforts it is fair to say that there are nevertheless drawbacks in trying to restrict oneself to pure refinement as the only way of going from an abstract to a concrete model, especially if the objective is to start from a simplified but transparent description, moving to a realistic but more complex description only by degrees. And size matters. Clearly there is little to be gained by presenting as small an example as the one above in this gradual manner, but one can imagine that in industrial scale situations, where there is much more complexity to manage, a multi stage development of a large specification is highly desirable, particularly if the size of the real specification is not dramatically smaller than that of the code that implements it, which can often happen when there is a lot of low level case analysis in the system description.

3 Retrenchment

In [1], we introduced retrenchment as a means of addressing the issues just highlighted, within a 'refinement-like' framework. In the context of B, a syntax very similar to that of the B REFINEMENT construct captures what is required, namely the following:

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	
$o \longleftarrow opname(i) \ \widehat{=}$		$p \longleftarrow opname(j) \ \widehat{=}$	
$S(u, i, o)$		BEGIN	

END

$T\,(\,v\,,j\,,p\,)$
[LVAR
A]
WITHIN
$P\,(\,i\,,j\,,u\,,v\,,A\,)$
CONCEDES
$C\,(\,u\,,v\,,o\,,p\,,A\,)$
END

END (3.1)

The left hand MACHINE $M(a)$, with operations $o \longleftarrow opname(i)$, each with body $S(u, i, o)$, is retrenched (via the RETRENCHES M clause and retrieve relation RETRIEVES $G(u, v)$), to MACHINE $N(b)$, with operations $p \longleftarrow opname(j)$. These latter operations now have bodies which are *ramified* generalised substitutions, that is to say generalised substitutions $T(v, j, p)$, each with its ramification, the LVAR , WITHIN and CONCEDES clauses. Each *opname* of M must appear ramified within N , but there can also be additional operations in N .

Speaking informally, the ramification of an operation allows us to describe how the concrete operation fails to refine its abstract counterpart. The optional LVAR A clause permits the introduction of logical variables A , that remember before-values of variables and inputs, should they be needed later. Its scope is the WITHIN and CONCEDES clauses. The WITHIN clause describes nontrivial relationships between abstract and concrete before-values of the state variables u and v , and abstract and concrete inputs i and j , and defines A if A is being used. It strengthens the precondition as we will see. The CONCEDES clause provides similar flexibility for the after-state, weakening the postcondition, and describes nontrivial relationships between abstract and concrete variables and abstract and concrete outputs, and using A if it has previously been defined.

The proof obligations make all this more precise. The conventional POs for the machines M and N hold, including the initialisation POs, $[\ X(u)\ ]\ I(u)$ and $[\ Y(v)\ ]\ J(v)$, and the machine invariant preservation POs, $I(u) \wedge \mathsf{trm}(S(u, i, o)) \Rightarrow [\ S(u, i, o)\ ]\ I(u)$ and $J(v) \wedge \mathsf{trm}(T(v, j, p)) \Rightarrow [\ T(v, j, p)\ ]\ J(v)$. A joint initialisation PO is also required to hold, being identical to the refinement case, $[\ Y(v)\ ] \neg [\ X(u)\ ] \neg\, G(u, v)$. Of most interest however is the retrenchment PO for operations which reads:

$$\begin{aligned}
&(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(T(v, j, p)) \wedge P(i, j, u, v, A)) \\
&\qquad \Rightarrow \\
&\quad \mathsf{trm}(S(u, i, o)) \wedge [\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg \\
&\qquad\quad (G(u, v) \vee C(u, v, o, p, A))
\end{aligned} \tag{3.2}$$

This contains on the left hand side the invariants $(I(u) \wedge G(u, v) \wedge J(v))$, and we strengthen the concrete trm predicate with $P(i, j, u, v, A)$, as stated above. The right hand side infers the abstract trm predicate, and the '$[\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg$' structure establishes in the after-states, the retrieve relation weakened by $C(u, v, o, p, A)$. A detailed heuristic discussion in [1] justified the shape of this PO. A major part of the purpose of this paper is to show how simulation properties support this choice.

We can now show how the low level machine *Mobile_Radio_LL* of the last section retrenches *Mobile_Radio_HL* . To conform to the above syntax we have to add the retrenchment declaration 'RETRENCHES *Mobile_Radio_HL*' , and of course the retrieves clause (2.4), but the main thing is the ramifications of the operations. We consider these in turn.

The *call_outgoing* operation can be dealt with as follows:

call_outgoing (*num*) $\widehat{=}$
BEGIN
PRE *bandSelected* = *TRUE* $\wedge$ *jcallState* = *Idle* $\wedge$ *num* $\in$ CHANELS
THEN
CHOICE *jcallState* := *Busy* || *jcurrChan* := *num*
OR *skip*
OR *jcallState* := *Jam*
END
END
LVAR
jCH
WITHIN
jCH = *jcurrChan*
CONCEDES
jcurrChan = *jCH* $\wedge$ *jcallState* = *Jam*
END

Note that the WITHIN clause serves only to define *jCH* . This is because the stronger concrete trm predicate appears as hypothesis in the operation PO, and the abstract one is deduced. Of course we could strengthen the WITHIN clause with '*bandSelected* = *TRUE*' to highlight the differences between the trm predicates if we wished. Note also how the CONCEDES clause captures what happens when the retrieve relation breaks down; here it is important to realise that the occurrence of the *jcurrChan* state variable in the WITHIN clause refers to its before-value, while the occurrence in the CONCEDES clause refers to its after-value, moreover *jCH* , being a fresh logical variable, remembers the former for use in the context of the latter.

The corresponding *call_incoming* operation needs only the trivial ramification WITHIN *true* CONCEDES *false* . This is because the trm predicates are the same for this case, and the '[$T(v, j, p)$] $\neg$ [$S(u, i, o)$] $\neg$' of the operation PO, requires the concrete guard to be stronger than the abstract guard (the same structure as found in refinement).

The remaining operations that need ramifying, *disconnect_outgoing* and *disconnect_incoming* can also be given the trivial ramification. As before this is because in retrenchment, both the guard and the termination predicate of the concrete version of an operation are required to be stronger than the abstract ones (pending qualification by the WITHIN clause). In this sense retrenchment has no preference between the 'called operation' and 'spontaneous event' views of an operation.

4 Stepwise Simulation

The semantic touchstone for retrenchment is stepwise simulation, by which we mean the simulation of a sequence of steps of the simulatee $\mathcal{T}$ by an equal length sequence of steps of the simulator $\mathcal{S}$, see Fig. 1. However the precise definition of 'simulates' in this context will depend on the precise relationship between the two systems under study so we will not give a formal definition here. Suffice for the moment to say that ‘simulates’ always includes the preservation of the retrieve relation, since there will always be one around.

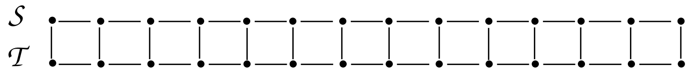

Fig. 1. A stepwise simulation.

We write a step of a machine such as M of (3.1) in the form:

u -(i, m, o)-> u'

where u and u' are the before and after states, m is the name of the operation (where it can help, we write S , the body of m , instead of m itself), and i and o are the input and output of m . This signifies that (u, i) satisfy $\mathsf{trm}(S)$, and that (u, i, u', o) satisfy the before-after predicate of m (this is the predicate $Q(u, u')$ in the normal form (2.1) when there is no I/O, otherwise it is $Q(u, i, u', o)$, the corresponding predicate for the normal form with I/O present). When discussing properties of sequences of steps, $\mathsf{last}(\mathcal{T})$ will denote the index of the last state mentioned in $\mathcal{T}$, and $r \in \mathrm{dom}^{\bullet}(\mathcal{T})$ will mean $r \in [0 \dots \mathsf{last}(\mathcal{T}) - 1]$ if $\mathcal{T}$ is finite, and $r \in \mathsf{NAT}$ otherwise. Similarly for sequences of any type. 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 $\mathsf{Ops}^M \subseteq \mathsf{Ops}^N$.

5 Modulated Refinement and Simulation

In this section we explore in some depth a notion, modulated refinement, that lies part way between conventional refinement and retrenchment, in order to illuminate the relationship between them. Modulated refinement is superficially a straightforward extension of refinement to allow different I/O signatures at the two levels of abstraction in question. In this sense part of what appears here bears comparison with other adaptations of refinement to cope with change of I/O representation such as [6, 7, 8, 9]. Here is a specific B syntax.

```
MACHINE            M ( a )

VARIABLES          u
INVARIANT          I ( u )

INITIALISATION     X ( u )
OPERATIONS
    o ⟵ opname ( i ) ≙
        S ( u , i , o )
END
```

```
MACHINE            N ( b )
MODREF             M
VARIABLES          v
INVARIANT          J ( v )
RETRIEVES          G ( u , v )
INITIALISATION     Y ( v )
OPERATIONS
    p ⟵ opname ( j ) ≙
        BEGIN
            T ( v , j , p )
        WITHIN
            P ( i , j , u , v )
        NEVERTHELESS
            V ( u , v , o , p )
        END
END                                   (5.1)
```

Note the MODREF keyword, indicating the intended relationship, and as in retrenchment, the separate appearance of the retrieve relation. Because modulated refinement mirrors conventional refinement, we demand that M and N have the same set of operation names Ops . In N , each operation has a WITHIN clause, but this time there are no logical variables, so the clause simply expresses the relationship between the before-states and the inputs, as an enhancement to the retrieve relation. Likewise there is a NEVERTHELESS clause, expressing the relationship between the after-states and the outputs. Unlike the CONCEDES clause of retrenchment, the NEVERTHELESS clause will act conjunctively, also enhancing the retrieve relation, hence the different name.

In fact the above syntax will serve for two slightly different notions of modulated refinement, *normal* and *inverted* to be introduced shortly, and we could introduce separate N-MODREF and I-MODREF keywords for these, but it is convenient not to do so. Until further notice we will study normal modulated refinement.

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

$$[\ Y(v)\ ] \neg [\ X(u)\ ] \neg \\ \qquad (G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\ \exists\, i_m \bullet P_m(i_m, j_m, u, v))) \tag{5.2}$$

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

$$(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(S(u, i, o)) \wedge P(i, j, u, v)) \\ \Rightarrow \\ \mathsf{trm}(T(v, j, p)) \wedge [\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg \\ \qquad (G(u, v) \wedge V(u, v, o, p)) \tag{5.3}$$

and lastly the operation compatibility PO, which for a typical operation n (where $n \in \mathsf{Ops}$, and has clauses P_n and V_n) reads:

$$G(u, v) \wedge V_n(u, v, o_n, p_n) \Rightarrow \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\ \exists\, i_m \bullet P_m(i_m, j_m, u, v)) \tag{5.4}$$

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 (similar remarks apply for (5.2)). Note the appearance in a conjunctive context of the NEVERTHELESS clauses in (5.3) and (5.4). The main reason for studying normal modulated refinement is that it posesses the natural analogue of the simulation property so characteristic of conventional refinement.

Definition 5.1 Let (5.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› v_1 -(j_1, m_1, p_2)-› $v_2 \ldots\ ]$ is a concrete execution sequence, and that $\mathcal{S} \equiv [\ u_0$ -(i_0, m_0, o_1)-› u_1 -(i_1, m_1, o_2)-› $u_2 \ldots\ ]$ 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 $r \in \mathrm{dom}^{\bullet}(\mathcal{T})$:

$$G(u_r, v_r) \wedge P_{m_r}(i_r, j_r, u_r, v_r) \wedge \\ \qquad G(u_{r+1}, v_{r+1}) \wedge V_{m_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}) \tag{5.5}$$

Definition 5.2 Let (5.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› v_1 -(j_1, m_1, p_2)-› $v_2 \ldots\ ]$ is a finite 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 :

$$\begin{aligned} \mathsf{trmP}_{M,r,r} &= \ \mathit{true} \\ &\dots\ \dots\ \dots \\ \mathsf{trmP}_{M,r,s} &= \ P_{m_r}(i_r, j_r, u, v_r) \wedge [\ S_{m_r}\ ]\ \mathsf{trmP}_{M,r+1,s} \end{aligned} \tag{5.6}$$

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

$$\mathsf{preP}_{M,ms} \ = \ \{(u_0, i_0, i_1 \dots i_{k-1}) \in \mathsf{U} \times \mathsf{I}_0 \times \mathsf{I}_1 \dots \times \mathsf{I}_{k-1} \mid \mathsf{trmP}_{M,0,z}\} \tag{5.7}$$

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

Theorem 5.3 Let (5.1) describe a normal modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-}\!\!> v_1 \text{ -}(j_1, m_1, p_2)\text{-}\!\!> v_2 \dots\ ]$, with sequence of invoked operation names $ms \equiv [\ m_0, m_1 \dots\ ]$, be a finite execution sequence of N . Suppose there is a $(u_0, i_0 \dots) \in \mathsf{preP}_{M,ms}$ such that u_0 also witnesses the initialisation PO (5.2). Then there is an execution sequence of M , $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-}\!\!> u_1 \text{ -}(i_1, m_1, o_2)\text{-}\!\!> u_2 \dots\ ]$, which is a stepwise simulation of $\mathcal{T}$.

Proof. Let $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-}\!\!> v_1 \dots\ ]$ 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 $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-}\!\!> u_1 \dots\ ]$ by an induction on $\mathrm{dom}^\bullet(\mathcal{T})$.

For $r = 0$, we know from the hypotheses that a $(u_0, i_0 \dots) \in \mathsf{preP}_{M,ms}$ exists such that $G(u_0, v_0)$ holds. Since $\mathsf{trmP}_{M,0,\mathsf{size}(ms)}(u_0, i_0 \dots) \Rightarrow \mathsf{trm}(S_{m_0})(u_0, i_0)$, we have $\mathsf{trm}(S_{m_0})(u_0, i_0) \wedge P_{m_0}(i_0, j_0, u_0, v_0) \wedge G(u_0, v_0)$. 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 $(u_0, i_0 \dots) \in \mathsf{preP}_{M,ms}$ we conclude that there is a $(u_1, i_1 \dots) \in \mathsf{preP}_{M,\mathsf{tail}(ms)}$, 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 the machine invariants $I(u_r)$ and $J(v_r)$ hold, and we also have[1] $G(u_r, v_r) \wedge (u_r, i_r \dots) \in \mathsf{preP}_{M,ms\downarrow r}$. This enables us to perform the inductive step as above. ☺

We cannot extend the above strategy to the case of infinite sequences $\mathcal{T}$, 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 5.4 Let (5.1) be a (normal or inverted) modulated refinement. Suppose $\mathcal{T} \equiv [\ v_0 \text{ -}(j_0, m_0, p_1)\text{-}\!\!> v_1 \text{ -}(j_1, m_1, p_2)\text{-}\!\!> v_2 \dots\ ]$ is a(n infinite) concrete execution sequence. Let

---

1. 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 $\mathrm{dom}(ms)$.

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

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

$$\forall\, r \in \mathrm{dom}^{\bullet}(\mathcal{T}) \bullet \mathrm{N}_r^{\mathsf{UI}} < \infty \tag{5.8}$$

Note that retrieve boundedness is inspired by the same thought as internal continuity in [21] and finite invisible nondeterminism in [22].

Theorem 5.5 Let (5.1) describe a normal modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› v_1 -(j_1, m_1, p_2)-› $v_2 \ldots\ ]$, 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 (5.2). Then there is an execution sequence of M , $\mathcal{S} \equiv [\ u_0$ -(i_0, m_0, o_1)-› u_1 -(i_1, m_1, o_2)-› $u_2 \ldots\ ]$, which is a stepwise simulation of $\mathcal{T}$.

Proof. Let $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› $v_1 \ldots\ ]$ be as given. We show there is a corresponding $\mathcal{S} \equiv [\ u_0$ -(i_0, m_0, o_1)-› $u_1 \ldots\ ]$ as follows. We know that each finite prefix of $\mathcal{T}$ can be stepwise simulated because of Theorem 5.3. 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 -(i_r, m_r, o_{r+1})-› 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 (5.3), the operation PO for inverted modulated refinement reads:

$$\begin{array}{l}(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(T(v, j, p)) \wedge P(i, j, u, v)) \\ \quad\Rightarrow \\ \quad \mathsf{trm}(S(u, i, o)) \wedge [\ T(v, j, p)\ ] \neg\ [\ S(u, i, o)\ ] \neg \\ \qquad (G(u, v) \wedge V(u, v, o, p))\end{array} \tag{5.9}$$

Note the inverted roles of the trm predicates.

Theorem 5.6 Let (5.1) describe an inverted modulated refinement where the common set of operation names is Ops . Let $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› v_1 -(j_1, m_1, p_2)-› $v_2 \ldots\ ]$ be an execution sequence of N . Then there is an execution sequence of M , $\mathcal{S} \equiv [\ u_0$ -(i_0, m_0, o_1)-› u_1 -(i_1, m_1, o_2)-› $u_2 \ldots\ ]$, which is a stepwise simulation of $\mathcal{T}$.

Proof. Let $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› $v_1 \ldots\ ]$ be an execution sequence of N . The $\mathrm{dom}(\mathcal{T}) = \{0\}$ case is as in Theorem 5.3. 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 $\mathcal{T}$, (5.2) 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. By the definition of execution step, $\mathsf{trm}(T_{m_0})(v_0, j_0)$ holds. Now the initialisation POs for M and N yield $I(u_0)$ and $J(v_0)$. Thus the operation PO (5.9), yields a step of M , u_0 -(i_0, m_0, o_1)-› 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 $\mathcal{S}$ 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 (5.4), we can infer that for the j_r and m_r given by $\mathcal{T}$, we can find an i_r such that $P_{m_r}(i_r, j_r, u_r, v_r)$ holds. This is enough to complete the inductive step as before. ☺

We can see clearly why we need different termination assumptions in Theorems 5.3 and 5.5 on the one hand, and Theorem 5.6 on the other. In the latter, we need a termination assumption about the steps of $\mathcal{T}$, a given execution sequence, to be able to exploit the operation PO. Since the individual concrete steps are each within their individual trm predicates, what we have is already sufficient. In the former, we need a termination assumption about the steps of $\mathcal{S}$, an execution sequence yet to be constructed. Therefore stronger assumptions are needed before the relevant operation PO can be used.

Theorem 5.6 can be understood as establishing a strong simulation in the automata theoretic sense, an intrinsically more local concept than the property in Theorem 5.3.

Definition 5.7 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:

$$v_0 \in \boldsymbol{V}_0 \ \Rightarrow \ \exists\, u_0 \in \boldsymbol{U}_0 \bullet u_0 \,\Theta_\mathrm{S}\, v_0 \tag{5.10}$$

and for all u , v :

$$u \,\Theta_\mathrm{S}\, v \ \wedge \ v \text{ -}\mu\text{-›}\, v' \in T_\mathrm{V} \ \Rightarrow \\ \exists\, u' \in \boldsymbol{U}, \lambda \in \boldsymbol{L}_\mathrm{U} \bullet u' \,\Theta_\mathrm{S}\, v' \ \wedge \ \lambda \,\Theta_\mathrm{L}\, \mu \ \wedge \ u \text{ -}\lambda\text{-›}\, u' \in T_\mathrm{U} \tag{5.11}$$

Note that Definition 5.7 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 5.8 Let M be a machine with operation names set Ops. Then the reachable transition system T_M associated to M is the initial state set $\boldsymbol{U}_0$ and subset T_M of $\boldsymbol{U} \times \boldsymbol{L}_M \times \boldsymbol{U}$ where:

(1) $\boldsymbol{L}_M = \{(i, m, o) \in \mathsf{I} \times \mathsf{Ops} \times \mathsf{O}\}$
(2) $\boldsymbol{U} = \{u \in \mathsf{U} \mid \text{there is an execution sequence of } M \text{ with } u \text{ as final state}\}$
(3) $\boldsymbol{U}_0 = \{u_0 \in \boldsymbol{U}^s \mid u_0 \text{ is an initial state of } M\}$
(4) $T_M = \{u \text{ -}(i, m, o)\text{-›}\, u' \mid \text{there is an execution sequence of } M$ with $u \text{ -}(i, m, o)\text{-›}\, u' \text{ as final step}\}$

Theorem 5.9 Let (5.1) describe an inverted modulated refinement where the common set of operation names is Ops. Then there is a strong simulation from the reachable transition system T_N of N to the reachable transition system T_M of M.

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

$$\Theta_\mathrm{S} = \{(u, v) \in \boldsymbol{U}^s \times \boldsymbol{V}^s \mid G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}} (\forall\, j_m\, \exists\, i_m \bullet P_m(i_m, j_m, u, v))\} \tag{5.12}$$

$$\Theta_\mathrm{L} = \{((i, m, o), (j, m, p)) \in (\mathsf{I} \times \mathsf{Ops} \times \mathsf{O}) \times (\mathsf{J} \times \mathsf{Ops} \times \mathsf{P}) \mid \\ (\exists\, u, v \bullet P_m(i, j, u, v)) \wedge (\exists\, u, v \bullet V_m(u, v, o, p))\} \tag{5.13}$$

Showing that this constitutes a strong simulation is a matter of routinely reworking the details of the inductive step of the proof of Theorem 5.6. ☺

We note that a strong simulation relates states to states, and transition labels to transition labels, essentially independently. In the case of Theorem 5.3, the properties demanded of abstract states at the beginning and end of a transition u -(i, m, o)-› u' , specifically that $(u, i \ldots) \in \mathsf{preP}_{M,m\to ms}$ 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 . Thus only the inverted case supports a proper notion of simulation. In contrast, in normal refinement (modulated or not), 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.

The preceding remarks depend on identifying the machine state with the automata theoretic state. If we relax this requirement, then we can incorporate history information in our notion of state, and then a notion of strong simulation can be recovered for the normal case (see eg. [21, 22, 23, 24, 25]), but this is a less natural correspondence than for the inverted case.

6 Simple Simulable Retrenchment

In this section we build on the insights of the preceding section to address the simulation and refinement properties of retrenchment. Now, the inclusion $\mathsf{Ops}^M \subseteq \mathsf{Ops}^N$ is generally a proper one. First we give the definition of stepwise simulation in this setting.

Definition 6.1 Let (3.1) be a retrenchment. Suppose that $\mathcal{T} \equiv [\ v_0$ -(j_0, m_0, p_1)-› v_1 -(j_1, m_1, p_2)-› $v_2 \ldots\ ]$ is an execution sequence of N , and that $\mathcal{S} \equiv [\ u_0$ -(i_0, m_0, o_1)-› u_1 -(i_1, m_1, o_2)-› $u_2 \ldots\ ]$ 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:

$$\begin{aligned} &G(u_r, v_r) \wedge P_{m_r}(i_r, j_r, u_r, v_r, A_r) \wedge \\ &\qquad (G(u_{r+1}, v_{r+1}) \vee C_{m_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}, A_r)) \end{aligned} \tag{6.1}$$

We now look at a simple sufficient condition for simulation and refinement. The properties of the relevant class of retrenchments are so strong that they are indeed almost refinements.

Definition 6.2 For a retrenchment like (3.1), suppose the joint initialisation establishes:

$$(G(u_0, v_0) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m\ \exists\, i_m, A_m \bullet P_m(i_m, j_m, u_0, v_0, A_m))) \tag{6.2}$$

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

$$\begin{aligned} &G(u, v) \vee C_n(u, v, o, p, B) \\ &\qquad \Rightarrow \\ &\quad (G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m\ \exists\, i_m, A_m \bullet P_m(i_m, j_m, u, v, A_m))) \end{aligned} \tag{6.3}$$

then we say that the retrenchment is a simple simulable retrenchment.

The proof of the following theorem is very similar to that of Theorem 5.6 (only the final part of the inductive step is changed slightly), and is given in full in [26].

Theorem 6.3 Let (3.1) describe a simple simulable 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 an execution sequence of N . Suppose that the sequence of invoked operation names $ms \equiv [\ m_0, m_1 \ldots\ ]$ is an Ops^M sequence. Then there is an execution sequence of M , $\mathcal{S} \equiv [\ u_0 \text{ -}(i_0, m_0, o_1)\text{-> } u_1 \text{ -}(i_1, m_1, o_2)\text{-> } u_2 \ldots\ ]$, which is a stepwise simulation of $\mathcal{T}$.

Theorem 6.3 leads directly to a strong simulation result analogous to Theorem 5.9 for simple simulable retrenchment.

Definition 6.4 Let M and N be machines with operation name sets Ops^M and Ops^N . The M-restricted reachable transition system of N (whose states are solely those reachable by 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) $\boldsymbol{V}_M = \{v \in \mathsf{V} \mid$ there is an execution sequence of N consisting solely of Ops^M operations, with v as final state$\}$
(3) $\boldsymbol{V}_{M0} = \{v_0 \in \boldsymbol{V}_M \mid v_0$ is an initial state of $N\}$
(4) $T_{N_M} = \{v \text{ -}(j, m, p)\text{-> } v' \mid$ there is an execution sequence of N consisting solely of Ops^M operations, with $v \text{ -}(j, m, p)\text{-> } v'$ as final step$\}$

Theorem 6.5 Let (3.1) describe a simple simulable retrenchment where the set of abstract operation names is Ops^M . Then there is a strong simulation from the M-restricted reachable transition system T_{N_M} of N to the reachable transition system T_M of M .

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

$$\Theta_S = \{(u, v) \in \boldsymbol{U} \times \boldsymbol{V}_M \mid G(u, v) \wedge \bigwedge_{m \in \mathsf{Ops}^M} (\forall\, j_m\, \exists\, i_m, A_m \bullet P_m(i_m, j_m, u, v, A_m))\}$$

$$\Theta_L = \{((i, m, o), (j, m, p)) \in (\mathsf{I} \times \mathsf{Ops}^M \times \mathsf{O}) \times (\mathsf{J} \times \mathsf{Ops}^M \times \mathsf{P}) \mid (\exists\, A \bullet (\exists\, u, v \bullet P_m(i, j, u, v, A)) \Leftarrow (\exists\, u, v \bullet C_m(u, v, o, p, A)))\}$$

☺

Simple simulable 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 6.6 Let (3.1) describe a simple simulable 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
$o \longleftarrow OpName(i) \ \hat{=}$	$p \longleftarrow OpName(j) \ \hat{=}$
$\quad S^{\forall T}(u, i, o)$	$\quad$ BEGIN
END	$\qquad T(v, j, p)$
	$\quad$ WITHIN
	$\qquad P^{\exists A}(i, j, u, v)$
	$\quad$ NEVERTHELESS
	$\qquad$ *true*
	$\quad$ END
	END $\qquad$ (6.4)

where:

$$S^{\forall T}(u, i, o) \equiv \text{PRE} \ (\forall v, j \bullet I(u) \wedge G(u, v) \wedge J(v) \wedge \mathsf{trm}(S(u, i, o)) \Rightarrow \mathsf{trm}(T(v, j, p))) \ \text{THEN} \ S(u, i, o) \ \text{END} \tag{6.5}$$

$$P^{\exists A}(i, j, u, v) \equiv (\exists A \bullet P(i, j, u, v, A)) \tag{6.6}$$

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[2] 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 (6.5) above.

We can now check that the POs of the simple simulable retrenchment imply those of the refinement. For the initialisation PO, reinterpreting the $\exists A$ in (6.2), yields (5.2) with $P_m^{\exists A}$ replacing the P_m. Likewise for the operation compatibility PO, weakening, and reinterpreting the $\exists A$ in (6.3), yields (5.4) 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:

$$(I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathsf{trm}(S^{\forall T}(u, i, o)) \wedge P^{\exists A}(i, j, u, v)) \Rightarrow \mathsf{trm}(T(v, j, p)) \wedge [T(v, j, p)] \neg [S^{\forall T}(u, i, o)] \neg G(u, v) \tag{6.7}$$

knowing that (3.2) and (6.3) hold. So let us hypothesise the antecedents of (6.7). 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 (6.7); 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 (3.2), so we infer in particular that:

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

Next we use the trm/prd version of the normal form for generalised substitutions given in [2] and elaborated to include I/O. Applying this to both T and S in (6.7), after a little manipulation we get:

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

$\mathsf{trm}(T) \wedge (\forall v\tilde{}\, p\tilde{} \bullet \mathsf{prd}(T) \Rightarrow \neg\, (\mathsf{trm}(S) \wedge (\forall u\tilde{}\, o\tilde{} \bullet \mathsf{prd}(S) \Rightarrow \ldots\ \ldots)))$

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

$\mathsf{trm}(T) \wedge (\forall v\tilde{}\, p\tilde{} \bullet \mathsf{prd}(T) \Rightarrow \neg\, (\mathsf{trm}(\Phi|S) \wedge (\forall u\tilde{}\, o\tilde{} \bullet \mathsf{prd}(\Phi|S) \Rightarrow \ldots\ \ldots)))$

Winding back the normal forms gives:

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

We use (6.3) on the $(G \vee C)$ term, and monotonicity, after which some simplification yields:

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

Now it remains to apply the deduction principle, to arrive at (6.7). We are done. ☺

The next two results follow from the preceding, the first by noticing that in the refinement operation PO, $\mathsf{trm}(T)$ may be hypothesised directly, and the second by noticing that the termination part of the PO becomes trivial.

Theorem 6.7 Let (3.1) describe a simple simulable retrenchment where the set of abstract operation names is Ops^M. Then (6.4), 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:

$$S^T(u, i, o) \equiv \text{PRE } \mathsf{trm}(T(v, j, p)) \text{ THEN } S(u, i, o) \text{ END} \tag{6.8}$$

and (6.6), describe a normal modulated refinement.

Theorem 6.8 Let (3.1) describe a simple simulable retrenchment where the set of abstract operation names is Ops^M. Then (6.4), with the occurrences of $M^{\forall T}$ and $S^{\forall T}$ replaced by M and S respectively, and (6.6), describe an inverted modulated refinement.

It is interesting to compare the three refinement results of Theorems 6.6, 6.7, 6.8, particularly with respect to the variables that occur in the various components. One thing we would like to do is to consider the various machines $M^{\forall T}$, M^T, M, that occur in these theorems as versions of the M that occurs in (3.1), since we would like the theorems to be saying something about the original simple simulable retrenchment (3.1). Clearly there is no problem in such an identification for Theorem 6.8, since the two versions of M are syntactically identical. However in the case of Theorems 6.6 and 6.7, there is more to be said. For Theorem 6.6, machines $M^{\forall T}$ and M involve the same free variables. Nevertheless one cannot say straightforwardly say that the two machines exist in the same universe, for the concrete variables v and j are mentioned in the precondition of $S^{\forall T}$, even though they are universally quantified away. What discloses the dependence of machine $M^{\forall T}$ on these concrete entities, is the free ocurrence of the relational (meta-)variables G, J, $\mathsf{trm}(T)$. These reveal that even though the concrete variables that they depend on are quantified away, whatever their identity (as concrete variables), they still take values in their own appropriate concrete universes. Thus one cannot regard $M^{\forall T}$ as a version of the M of (3.1) without taking this amplification of the universe of variable values into account. Of course this is not a novel phenomenon since it is familiar already from standard B refinement. (See [2] Ch. 11, where a refinement of an abstract machine, though free only in the concrete variables, is nevertheless viewed as a 'differential' added to the abstract machine, this being expressed via the existential quantification of the abstract variables in the concrete construct).

The issue we have mentioned strikes us even more forcefully in the case of Theorem 6.7, where the concrete variables actually occur free in the machine M^T via (6.8). Here we cannot pretend that M^T exists in the same world as M ; the applicability of an abstract operation of M^T depends on the values of concrete variables that we wish to relate to the abstract variables at the point of application. Still, the discusion above persuades us that the difference between the cases of Theorems 6.6 and 6.7 is not as great as it might at first appear to be.

7 The Mobile Radio Example Revisited

As given, our mobile radio example does not in fact provide us with a simple simulable retrenchment. The problem lies with the *call_outgoing* operation, because it can violate the retrieve relation, as revealed in Section 3, where the possibility *jcallState* = *Jam* admitted in the CONCEDES clause conflicts with the retrieve relation clause *jcallState* = *callState*. However it is not hard to prove that we would have had a simple simulable retrenchment if we had substituted the *call_outgoing* operation in *Mobile_Radio_LL* with the following more robust version in which the *Jam* state is not a possible outcome:

call_outgoing (*num*) $\triangleq$
    PRE *bandSelected* = *TRUE* $\wedge$ *jcallState* = *Idle* $\wedge$ *num* $\in$ CHANELS
    THEN
        CHOICE *jcallState* := *Busy* || *jcurrChan* := *num*
        OR *skip*
        END
    END

This version of the operation requires only the trivial ramification to remain within the retrenchment. Note that even though the resulting system cannot violate the retrieve relation via any of the retrenched operations, such violation can still take place via the *fadeout* event which leads to *jcallState* = *Jam*. Thus even the simple simulable case of retrenchment expresses things that eg. superposition refinement cannot.

Of course in reality, the situation described in *Mobile_Radio_LL* will be the more typical one. Operations will have the capacity to yield results which either obey the retrieve relation or not. Those calls of the operation that remain within the retrieve relation will be simulable, the others not. A theoretically weaker framework will be able to distinguish between these cases, to derive stepwise simulations of the well behaved execution sequences. However the price paid for the presence of the others is the absence of results like Theorems 6.6, 6.7, 6.8 which involve an implicit quantification over all possible cases. The details are beyond the scope of this paper.

8 Conclusions

In this paper we started by looking at the passage from a relatively simple description of the key elements of a system, to a more comprehensive and therefore more cluttered picture, encompassing much detail that 'could at the start be left till later'. This kind of piecemeal buildup is typical of what goes on in realistic system design in its initial and preformal stages. Leaving details till later is usually not a symptom of laziness, but a pragmatic response to the task of understanding the complexity of a large system. In discussing our example we debated the extent to which existing elaborations of refinement were capable of giving an account of this process and we concluded that none of the existing ones fully covered what was needed. Our retrenchment proposal allowed the inclusion of the desired detail in a flexible framework that also incorporated the most pertinent aspects of existing techniques. One benefit of retrenchment is that it allows the controlled denial of previously stated abstract properties. This is useful since a system structuring strategy that totally forbids such denial can force the system structure into a form that appears upside down when compared with normal engineering intuition. Consider a simple example: if one were not allowed to contradict the unrealistically unbounded nature of Peano natural numbers, then strictly speaking, the properties of finite arithmetic ought to come out as the top level component of almost *any* design that requires calculations.

Our main concern in this paper was to explore some of the semantic properties of retrenchment, focusing on simulation. Our approach bears comparison with similar work eg. [27, 28]. In our case we introduced modulated refinement as a notion intermediate between conventional refinement and retrenchment. A key observation was that in the inverted version of this concept, we could relate sequence-oriented simulation and automata theoretic strong simulation whilst making a natural identification of the notions of state involved, a situation that fails for conventional refinement. We then applied these insights to a simple special case of retrenchment, the simple simulable case. We consider the simulation theoretic properties exhibited by this special case, (and others, based on weaker assumptions, whose treatment is beyond the scope of this paper), as ample retrospective reinforcement of the semantics of retrenchment given in the proof obligations proposed in [1] on purely heuristic grounds. Essentially, when models are connected only weakly as in retrenchment, a more unidirectionally oriented relationship between them is more informative than the more intimately interdependent one expressed through (normal) refinement. Even so, for the simple simulable special case, we were able to recover a notion of refinement, albeit at the price of some frame issues.

References

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

2. Abrial J. R. The B-Book. Cambridge University Press, 1996.

3. Wordsworth J. B. Software Engineering with B. Addison-Wesley, 1996.

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

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

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

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

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

9. Stepney S., Cooper D., Woodcock J. More Powerful Z Data Refinement: Pushing the State of the Art in Industrial Refinement. *in*: Proc. ZUM-98, Bowen, Fett, Hinchey (eds.), Springer, 1998, 284-307, LNCS **1493**.
10. Back R. J. R., Kurki-Suonio R. Decentralisation of Process Nets with Centralised Control. *in*: Proc. 2nd ACM SIGACT-SIGOPS Symp. on Princ. Dist. Comp., 131-142, ACM, 1983.
11. Back R. J. R. Refinement Calculus Part II: Parallel and Reactive Systems. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), Springer, 1989, 67-93, LNCS **430**.
12. Back R. J. R., von Wright J. Trace Refinement of Action Systems. *in*: Proc. CONCUR-94, Jonsson, Parrow (eds.), Springer, 1994, 367-384, LNCS **836**.
13. Francez N., Forman I. R. Superimposition for Interactive Processes. *in*: Proc. CONCUR-90, Baeten, Klop (eds.), Springer, 1990, 230-245, LNCS **458**.
14. Katz S. A Superimposition Control Construct for Distributed Systems. ACM Trans. Prog. Lang. Sys. **15**, 337-356, 1993.
15. Back R. J. R., Sere K. Superposition Refinement of Reactive Systems. Form. Asp. Comp. **8**, 324-346, 1996.
16. Blikle A. The Clean Termination of Iterative Programs. Acta Inf. **16**, 199-217, 1981.
17. Coleman D., Hughes J. W. The Clean Termination of Pascal Programs. Acta Inf. **11**, 195-210, 1979.
18. Neilson D. S. From Z to C: Illustration of a Rigorous Development Method. PhD. Thesis, Oxford University Computing Laboratory Programming Research Group, Technical Monograph PRG-101, 1990.
19. Owe O. 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, 1985.
20. Owe O. Partial Logics Reconsidered: A Conservative Approach. Form. Asp. Comp. **3**, 1-16, 1993.
21. Jonsson B. Simulations between Specifications of Distributed Systems. *in*: Proc. CONCUR-91, Baeten, Groote (eds.), Springer, 1991, 346-360, LNCS **527**.
22. Abadi M., Lamport L. The Existence of Refinement Mappings. Theor. Comp. Sci. **82**, 253-284, 1991.
23. Jonsson B. On Decomposing and Refining Specifications of Distributed Systems. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), Springer, 1989, 361-385, LNCS **430**.
24. Lynch N. Multivalued Possibilities Mappings. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), Springer, 1989, 519-543, LNCS **430**.
25. Merritt M. Completeness Theorems for Automata. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), Springer, 1989, 544-560, LNCS **430**.
26. Banach R., Poppleton M. Retrenchment and Punctured Simulation. *in*: Proc. IFM-99, Taguchi, Galloway (eds.), 457-476, Springer, 1999.
27. Derrick J., Bowman H., Boiten E., Steen M. Comparing LOTOS and Z Refinement Relations. *in*: Proc. FORTE/PSTV-9, 501-516, Chapman and Hall, 1996.
28. Bolton C., Davies J., Woodcock J. On the Refinement and Simulation of Data Types and Processes. *in*: Proc. IFM-99, Taguchi, Galloway (eds.), 273-292, Springer, 1999.