ISSN 1361 - 6161

THE UNIVERSITY of MANCHESTER

Computer Science
University of Manchester

Retrenchment: An Engineering Variation on Refinement

R. Banach, M. Poppleton

Department of Computer Science
University of Manchester
Technical Report Series
UMCS-99-3-2

Retrenchment: An Engineering Variation on Refinement*

R. Banach, M. Poppleton

Department of Computer Science
University of Manchester
Oxford Road, Manchester, U.K.
banach@cs.man.ac.uk, m.r.poppleton@coventry.ac.uk

1 March 1999

---

* Refereed by: Graham Gough.

Retrenchment: An Engineering Variation on Refinement

R. Banach[a], M. Poppleton[a,b]
[a]Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
[b]School of Mathl. and Inf. Sciences, Coventry University, Coventry, CV1 5FB, U.K.
banach@cs.man.ac.uk , m.r.poppleton@coventry.ac.uk

Abstract: It is argued that refinement, in which I/O signatures stay the same, preconditions are weakened and postconditions strengthened, is too restrictive to describe all but a fraction of many realistic developments. An alternative notion is proposed called retrenchment, which allows information to migrate between I/O and state aspects of operations at different levels of abstraction, and which allows only a fraction of the high level behaviour to be captured at the low level. This permits more of the informal aspects of design to be formally captured and checked. The details are worked out for the B-Method.

Keywords: Refinement, Retrenchment, Specification, Formal Development, B Method.

1 Idealised and Realistic Modelling: The Inadequacy of Pure Refinement

Like all good examples of terminology, the word "refinement" is far too evocative for its use ever to have been confined to exactly one concept. Even within the formal methods community, the word is used in at least two distinct senses. The first is a strict sense. An operation O_C is a refinement of an operation O_A iff the precondition of O_C is weaker than the precondition of O_A and the relation of O_C is less nondeterministic than the relation of O_A . The well known refinement calculus [Back (1981), Back (1988), Back and von Wright (1989), von Wright (1994), Morris (1987), Morgan (1990)] captures this in a formal system within which one can calculate precisely.

However there is a second, much less strict use of the word. In formalisms such as Z or VDM [Spivey (1993), Hayes (1993), Jones (1990), Jones and Shaw (1990)], requirements are frequently captured at a high level of abstraction, often involving for instance *divine* natural numbers or *divine* real numbers[1], and neglecting whole rafts of detail not appropriate to a high level view, in order that the reader of the high level description "can see the wood for the trees". Such descriptions are then "refined" to lower levels of abstraction where the missing details are filled in, typically yielding longer, more tortuous and much less transparent but much more realistic definitions of the system in question. Indeed the complexity of such descriptions can often be comparable to or greater than that of their implementations, a fact cited by detractors of formal methods as undermining the value of formal methods themselves, though this seems to us to be like denigrating stereoscopic vision because the image seen by the left eye is of comparable complexity to that seen by the right.

1. By *divine* naturals, integers or reals, we mean the natural numbers, integers or real numbers that God made, abstract and infinite, in contrast to the finite discrete approximations that we are able to implement on any real world system. The latter we call *mundane* natural numbers, integers or real numbers.

In truth the world is a complex place and developing descriptions of some part of it in two distinct but reconcilable formalisms (the specification and implementation), rather than just one, is always likely to help rather than hinder, even if the "more abstract" description is not significantly simpler than the other. (For an entertaining example of the undue weight attatched to the brevity of descriptions see Fig. 16.2 of [Wand and Milner (1996)].)

In this paper we address the main problem posed by the second use of the "refinement" word, namely that there is not a refinement relationship in the strict sense between the idealised high level specification, and the "real" but lower level specification. We will use the B framework throughout the paper, but will frequently pretend that B contains many more liberal types, à la Z or VDM , than it actually does. For instance we will assume available to us divine types such as $\mathsf{D\text{-}NAT}$, $\mathsf{D\text{-}INT}$, $\mathsf{D\text{-}REAL}$ as well as their mundane counterparts $\mathsf{M\text{-}NAT}$, $\mathsf{M\text{-}INT}$, $\mathsf{M\text{-}FLOAT}$; we can identify $\mathsf{M\text{-}NAT}$ with the normal B type of `NAT` .

Let us illustrate with a small example, namely addition. In negotiating the requirement for a proprietary operation with a customer we might write:

```
MACHINE          My_Divine_Machine
VARIABLES        aa , bb , cc
INVARIANT        aa ∈ D-NAT ∧ bb ∈ D-NAT ∧ cc ∈ D-NAT
    … …
OPERATIONS
    MyPlus ≙ cc := aa + bb ;
    … …
```

In $\mathsf{D\text{-}NAT}$, the + operation is the familiar one given by (say) the Peano axioms for addition, and is an ideal and infinite operation, making *MyPlus* equally ideal, but allowing us to see the essence of what is required. Having assured ourselves and the customer that we were on the right lines, we would want to describe more precisely what we could achieve, writing say:

```
MACHINE          My_Mundane_Machine
    … …
VARIABLES        aaa , bbb , ccc
INVARIANT        aaa ∈ M-NAT ∧ bbb ∈ M-NAT ∧ ccc ∈ M-NAT
    … …
OPERATIONS
    resp ⟵ MyPlus ≙
        IF
            aaa + bbb ≤ MaxNum
        THEN
            ccc := aaa + bbb ||
            resp := TRUE
        ELSE
            resp := FALSE
        END ;
    … …
```

My_Mundane_Machine could never be a refinement of *My_Divine_Machine* . Partly this is for trivial syntactic reasons, eg. we would have to write REFINEMENT

My_Mundane_Machine REFINES *My_Divine_Machine* …. Apart from that, there are three further important issues. Firstly the INVARIANT of the more concrete machine does not contain any refinement relation that would relate the abstract and concrete variables. This could easily be fixed if we were to write in the concrete INVARIANT say:

INVARIANT $\quad aaa \in \mathsf{M\text{-}NAT} \wedge bbb \in \mathsf{M\text{-}NAT} \wedge ccc \in \mathsf{M\text{-}NAT} \wedge$
$\quad aa = aaa \wedge bb = bbb \wedge cc = ccc$

assuming the obvious theory that allowed the identification of the mundane naturals as a subset of the divine ones. Secondly the signatures of *MyPlus* in the two machines are different; this is not allowed in normal notions of refinement. Thirdly, since not all divine naturals are refined to mundane ones, the standard proof obligation of refinement:

$$PA_{\mathrm{A,C}} \wedge INV_{\mathrm{A}} \wedge INV_{\mathrm{C}} \wedge \mathsf{trm}(MyPlus_{\mathrm{A}})$$
$$\Rightarrow \mathsf{trm}(MyPlus_{\mathrm{C}}) \wedge [MyPlus_{\mathrm{C}}] \neg [MyPlus_{\mathrm{A}}] \neg INV_{\mathrm{C}}$$

cannot possibly be satisfied, as the $aaa + bbb > MaxNum$ situation yields incompatible answers in the divine and mundane cases. (In the preceding the A and C subscripts refer to abstract and concrete respectively, $\mathsf{trm}(S)$ is the predicate under which operation S is guaranteed to terminate, and $PA_{\mathrm{A,C}}$ is the usual collection of clauses about the parameters and constants of the two machines. Furthermore $[MyPlus_{\mathrm{C}}] \neg [MyPlus_{\mathrm{A}}] \neg INV_{\mathrm{C}}$ says that for all steps that $MyPlus_{\mathrm{C}}$ might make, there is a step that $MyPlus_{\mathrm{A}}$ can make, that establishes INV_{C} , which contains the retrieve relation as noted above.)

The latter two reasons in particular show that the primary motivation for classical refinement, i.e. that the user should not be able to tell the difference between using the abstract or concrete version of an operation, does not hold sway here. What we are doing is adding real world detail to a description in a disciplined manner in order to aid understandability, not performing an implementation sleight of hand that we do not intend the user to notice.

Sometimes the process of adding detail can nevertheless be captured, albeit perhaps inelegantly, within the classical notion of refinement. For example the following would be a valid refinement if D-NAT were a valid type in B:

MACHINE $\quad$ *Your_Divine_Machine*
VARIABLES $\quad aa\ , bb\ , cc$
INVARIANT $\quad aa \in \mathsf{D\text{-}NAT} \wedge bb \in \mathsf{D\text{-}NAT} \wedge cc \in \mathsf{D\text{-}NAT}$
… …
OPERATIONS
$\quad YourPlus \ \hat{=}\ \ cc := aa + bb\ \ [\,]\ \ skip\ \ ;$
… …
END

REFINEMENT $\quad$ *Your_Mundane_Machine*
REFINES $\quad$ *Your_Divine_Machine*
… …
VARIABLES $\quad aaa\ , bbb\ , ccc$
INVARIANT $\quad aaa \in \mathsf{M\text{-}NAT} \wedge bbb \in \mathsf{M\text{-}NAT} \wedge ccc \in \mathsf{M\text{-}NAT} \wedge$
$\quad aa = aaa \wedge bb = bbb \wedge cc = ccc$
… …
OPERATIONS
$\quad YourPlus \ \hat{=}$

```
        IF
            aaa + bbb ≤ MaxNum
        THEN
            ccc := aaa + bbb
        END ;
    … …
END
```

Thus as long as the extra detail is hidden at the lower level, one can conceal a certain amount of low level enrichment by specifying *skip* as the whole or an optional part of the higher level definition. This is frequently done in B when what one really wants to do is to give a detailed description at (say) the implementation level, but one nevertheless needs a specification of one sort or another because the B method always demands a machine at the top level of a development. Under such circumstances one writes *skip* at the top level (usually omitting to define any abstract variables too), relying on the fact that any operation (whose effect on the concrete variables does not entail any visible consequences on the abstract variables via the refinement invariant) refines it. In effect one simply avoids the issue.

However we argue that such uses of *skip* , though technically neat where they can accomplish what is desired, are somewhat misleading. They mix concerns in the following sense. The job of the abstract specification is to set out the idealised model as clearly as possible. Matters are therefore not helped by occurences here and there of *skip* , whose purpose is to mediate between the abstract model and concrete model in order that the relationship between them should be a refinement in the strict sense. In fact the *skip* is signalling that the relationship between the abstract and concrete models is *not* one of pure refinement, but something more intricate. And thus a cleaner design strategy would place the data that described this relationship in an appropriate position, rather than clog up the abstract model with *skip*s.

Situations much more involved than either of the above can arise routinely in the development of certain types of critical system; in particular if the system in question must model phenomena described in the real world by continuous mathematics. In such cases, system construction may well be founded upon a vast aggregate of conventional mathematics, perhaps supported by semiempirical considerations, and having an accuracy requirement for calculations measured in percent rather than in terms of the exact embedding of M-FLOAT in D-REAL . In such cases, the derivation of the discrete model actually cast in software from the original high level model, is a complex process of reasoning steps, some more precise than others, and all in principle belonging in the safety case for the implemented system, as justification for the purported validity and range of applicability of the low level discrete model. In these cases, current refinement technology cannot speak about any but the last step or two, as the premise on which it is founded, namely that the user should not be able to tell if it is the abstract or concrete version of a system that he is using, is neither applicable nor relevant to most of the justification.

The use of *skip* to circumvent the gap between levels of abstraction is even less convincing than previously when starting from a continuous model. Suppose one is modelling Newton's Second Law, which equates acceleration with force divided by mass. Newton did not state "$a := f \,/\, m$ [] *skip*" or anything similar, and the interpolation of *skip*s in the statement of such continuous laws for the purposes stated is particularly unattractive and intrusive.

And yet it is very unsatisfying to say that the corroboration that formal methods can offer in critical developments should be abandoned in such cases because the nature of the reasoning involved is “out of scope” of the conventional notion of refinement. Rather we should look to enrich what we might accomplish by “refinement-like” concepts in the hope of bringing more useful engineering within the remit of formal methods. As the pressure to include software in more and more critical systems increases, and the pressure to verify such systems mechanically against state of the art methodology increases too, the market for such enrichments can only grow. Our conclusion then is that there is an identifiable need to search for more flexible ways of “refining” abstract requirements into implementable specifications. In Section 2 we introduce a liberalisation of refinement, retrenchment, as a step in this direction. Section 3 makes the proposal more precise by discussing the incorporation of retrenchment in the B-Method, while Section 4 looks at scenarios involving the passage from continuous to discrete mathematics, scenarios that are gaining importance in the serious application of formal methods to real world safety critical systems; places indeed where the notion of retrenchment is particularly likely to prove useful. Section 5 concludes. An earlier version of this paper appeared in the proceedings of the Second International B Conference [Bert (1998)].

2 Retrenchment

A refinement as everyone knows, weakens the precondition and strengthens the postcondition of an operation. Since an operation is specified by a precondition/postcondition pair, to go beyond refinement to relate an abstract operation O_{A} and a concrete operation O_{C} , either the precondition of O_{C} must be stronger than or unrelated to the precondition of O_{A} , or the postcondition of O_{C} must be weaker than or unrelated to the postcondition of O_{A} , or both. There are no other possibilities.

Retrenchment is, very loosely speaking, the strengthening of the precondition and the weakening of the postcondition though technically it’s more subtle than that. This is like the opposite of refinement, except that we avail ourselves of the opportunity to liberalise the connection between abstract and concrete operations even more widely. For instance not only will we allow changes of data type in the state component of an operation, we will also allow flexibility in the input and output components. Thus we allow inputs and outputs to change representation between abstract and concrete operations, and moreover we allow information to drift between I/O and state aspects during a retrenchment. Thus some data that was most conveniently viewed as part of the input at the abstract level say, might be best recast as partly input data and partly state at a more concrete level, or vice versa. Similar things might occur on the output side. This greater flexibility in involving properties of the inputs and outputs in the relation between versions of an operation, gives more leeway for building realistic but complex specifications of real systems out of oversimplified but more comprehensible subsystems.

These things go way beyond what is conceivable in refinement. Correspondingly, the usual way of controlling refinement, via a joint invariant, will be inadequate as a means of expressing the properties of this more liberal situation. We will need to split up the “local invariant” and “retrieve relation” in the joint invariant, and a couple of extra predicates per operation, one for the before-aspect and one for the after-aspect, and if it is necessary to relate the two in a general manner, a means of declaring logical variables with both of these predicates as scope. We see this in detail in the next section.

Of course this means that the litmus test of conventional refinement, that the correspondence between states at different levels of abstraction, and the identity of the inputs and outputs of corresponding operation instances, can be extended to a similar correspondence for sequences of operation applications, no longer applies if information can be moved between the I/O and state aspects of an operation in a retrenchment. When such a state of affairs holds, then there will usually be additional properties of histories of the two systems in question that are of interest in the overall understanding of the problem being solved. These sequence-oriented properties might typically include liveness and other fairness properties [see eg. Abadi and Lamport (1991)]. We do not pursue these aspects in the present paper, any more than fairness properties are covered by conventional formal methods of the model oriented kind, though further work may reveal the desirability of doing so.

It must be borne in mind that despite the similarities in the terminology in which it is phrased, the retrenchment technique proposed here is intended to be regarded in an entirely different light than conventional refinement. Refinement has as its objective a sort of black box role. The user does not need to enquire into the contents of the refinement black box, and the mathematical soundness of the notion of refinement guarantees that he is never wrong-footed by not doing so, since the observable behaviour of a refinement is always a possible behaviour of the abstraction. Retrenchment on the other hand, has as its objective very much a white box role. Retrenchment is a conscious decision to solve a different problem, and this must always be a deliberate engineering decision, deemed acceptable under prevailing circumstances. We envisage that the extent to which any particular retrenchment step can be justified on entirely self-contained mathematical grounds will very much vary from application to application; we imagine most will not be able to be so justified without input from non-mathematically-derivable real world considerations. (For example, mathematics can *express* the fact that the mundane naturals are finite, but it cannot *derive* this fact from some convincingly self-evident abstract criteria.)

3 Incorporating Retrenchment in the B-Method

The B-Method [Abrial (1996), Lano and Haughton (1996), Wordsworth (1996)] is a semantically well founded and structurally rich methodology for full-lifecycle formal software development. As such it provides an ideal framework into which to embed the retrenchment concept, since it already provides syntactic structure for expressing the refinement relation between specifications and implementations. Retrenchment will need a mild generalisation of this.

3.1 Syntax for Retrenchment

We propose the following outline syntax for retrenchment constructs where the square brackets indicate that LVAR *A* is optional:

MACHINE	*Concrete_Machine_Name* (*params*)
RETRENCHES	*Abstract_Machine-or-Refinement_Name*
… …	
INVARIANT	*J*

```
RETRIEVES          G
    … …
OPERATIONS
    out ⟵ OpName ( in ) ≙
        BEGIN T [ LVAR A ] WITHIN P CONCEDES C END ;
    … …
END
```

In the above, we propose that the retrenchment construct be a MACHINE, as a retrenchment, being a decision to solve a new problem, should have a MACHINE as its top level statement. The RETRENCHES clause relates the retrenching machine to the retrenched construct that it remodels. We propose that the latter be either a MACHINE or a REFINEMENT for maximum flexibility. The RETRENCHES clause is similar to the REFINES clause in standard B, in that it opens the retrenched construct and makes its contents visible in the retrenching machine for the purpose of building predicates involving the contents of both. We assume that the name spaces of retrenching and retrenched constructs are disjoint aside from the operation names, which must admit an injection from operation names of the retrenched construct to operation names of the retrenching machine. (The reason we do not demand a bijection is that, given that we wish to allow the retrenched construct to be a proper oversimplification of the retrenching machine, there may well be operations of the retrenching machine that do not make sense at the more abstract level of the retrenched construct. Such operations could be modelled at the abstract level by *skip*s of course, but we would just as soon not do so. Having an injection instead of a bijection on operation names can lead to interesting repercussions at the level of simulation, as indicated below.)

The retrenching machine can be parameterised (in contrast to refinements), so the CONSTRAINTS clause of the retrenching machine becomes in principle a joint predicate involving parameters of both retrenching and retrenched constructs if there is a need to express a relationship between them.

Being a machine, we propose that all the familiar machine structuring facilities: INCLUDES, USES, SEES, PROMOTES, EXTENDS, are available to the retrenching machine in the normal way. These aspects of machine construction are orthogonal to the retrenching idea.

Like machines but unlike refinements, the INVARIANT clause of a retrenching machine is a predicate in the local state variables only. Joint properties of state variables of both retrenching and retrenched constructs are held in the RETRIEVES clause, as they need to be treated a little differently compared to refinements.

The main difference between ordinary machines and retrenching machines appears in the operations. We propose to call the body of an operation of a retrenching machine a *ramified generalised substitution*. A ramified generalised substitution BEGIN T LVAR A WITHIN P CONCEDES C END , consists of a generalised substitution T as for any normal operation, together with its ramification, which consists of the following: the LVAR A clause which can declare logical variables A whose scope is both the WITHIN P and CONCEDES C clauses; the WITHIN P clause which is a predicate that defines the logical variables A and can strengthen the precondition of T by involving the abstract state and input; and the CONCEDES C clause which is a predicate that can weaken the postcondition of T by involving the abstract state and output and the logical variables A . Readers who desire more intuition at this point may profitably look at the simple examples of retrenchment in section 3.4 now.

As mentioned in the previous section, global properties of system histories might well be expected to play a more significant role in a retrenchment than is usually the case in a refinement. There may thus be good reason to include a "SIMULATION Θ" clause in a retrenchment construct, where Θ describes a relationship between sets of sequences of operations at the two levels of abstraction, and including where appropriate relationships between values of input and output for corresponding operation instances. However in the general case, one might well have to incorporate properties of the system's environment, and so we do not pursue such a possibility in this paper.

3.2 Proof Obligations for Retrenchment

In order to discuss the proof obligations for retrenchment in detail, consider the following two machines. For simplicity we will assume that these machines contain no ASSERTIONS or DEFINITIONS which could be dealt with in the standard way.

```
MACHINE          M ( a )
 
VARIABLES        u
INVARIANT        I ( u )
 
INITIALISATION   X ( u )
OPERATIONS
    o <--- OpName ( i ) ≙
        S ( u , i , o )
END
```

```
MACHINE          N ( b )
RETRENCHES       M
VARIABLES        v
INVARIANT        J ( v )
RETRIEVES        G ( u , v )
INITIALISATION   Y ( v )
OPERATIONS
    p <--- OpName ( j ) ≙
        BEGIN
            T ( v , j , p )
        LVAR
            A
        WITHIN
            P ( i , j , u , v , A )
        CONCEDES
            C ( u , v , o , p , A )
        END
END
```

For these machines there will be the usual machine existence proof obligations (unless one postpones them till machine instantiation time as in standard B). Moreover, we point out that if the CONSTRAINTS clause of the retrenching machine is a joint predicate, then if there are a number of retrenchments and refinements in a development, the CONSTRAINTS proof obligations will grow to encompass all of them simultaneously, as $\exists x.P \wedge \exists x.Q \not\Rightarrow \exists x.(P \wedge Q)$; i.e. the values that witness joint machine existence in two adjacent retrenchment steps need not be the same ones for the middle machine. We do not foresee this as a major difficulty as a realistic development is unlikely to contain very many retrenchment steps.

There will be standard operation proof obligations viewing both M and N as machines in isolation. These include the initialisation proof obligations:

$$PA_M \Rightarrow [\ X(u)\ ]\ I(u)$$

$$PA_N \Rightarrow [\ Y(v)\ ]\ J(v)$$

The operation invariant proof obligations are as usual:

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

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

There is also a standard "refinement style" obligation to prove that both initialisations establish the retrieve relation G , i.e. that:

$$PA_{M,N} \Rightarrow [\ Y(v)\ ] \neg [\ X(u)\ ] \neg\ G(u, v)$$

The most interesting proof obligations are the retrenchment ones. For a typical abstract operation *OpName* given abstractly by S and retrenched to a ramified operation given by BEGIN T … , this reads as follows.

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

We regard this statement as a definition of the semantics of retrenchment, as opposed to refinement where the corresponding statement in Section 1 supports a more abstract formulation, based upon set transformer or relational inclusion etc., (see eg. Abrial (1996) Chapter 11). We base our definition on the following.

Let us reexamine refinement for a moment. In refinement, the objective is to ensure that the concrete system is able to emulate the abstract system, and in general there will be a many-many relationship between the steps that the abstract and concrete systems are able to make such that this is true. In particular, whenever an abstract operation is ready to make a terminating step, the corresponding concrete operation must be prepared to make a terminating step, which is the first conjunct of the refinement proof obligation. Furthermore, whenever a concrete operation actually makes a step, the result must be not incompatible with some step that the corresponding abstract operation could have made at that point. For this it is sufficient to exhibit for every concrete step, an appropriate abstract step: the second conjunct. Thus the $\forall$*Conc-Op*$\exists$*Abs-Op*… structure of the second conjunct comes from ensuring that no concrete step "does anything wrong".

Retrenchment is different since the abstract and concrete systems are definitely incompatible. The white box nature of retrenchment implies that the relationship between abstract and concrete systems should be viewed first and foremost as an enhancement to the description of the concrete system, for that is the purpose of retrenchment. A retrenchment proof obligation ought to reflect this.

As before, there will in general be a many-many relationship between those steps that the abstract and concrete systems are able to make, that we might want to regard as related. Since in a retrenchment, it is the more concrete system that is considered more important, in the hypotheses of the proof obligation, it is the concrete trm condition for an operation that is present. We strengthen this by the WITHIN clause P to take into account aspects arising from the abstract state, abstract and concrete inputs, and to allow further fine tuning of the related before-configurations above and beyond that given by the RETRIEVES clause G , if required.

What then ought the conclusions of such a proof obligation to assert? The trm condition for the corresponding abstract operation is the obvious first thing. And the obvious second thing would speak about results in related steps. For these we would require the truth of the RETRIEVES clause G , but weakened by the CONCEDES clause C to take into account aspects arising from the abstract state, abstract and concrete output, and to allow deviations from strictly "refinement-like" behaviour to be expressed. We must say which pairs of abstract and concrete after-configurations should be ($G \vee C$)-related in the proof obligation. The essentially arbitrary nature of the many-many relation between steps makes expressing it verbatim within the proof obligation impractical and certainly not mechanisable. We are left with the possibility of stating some stylised subrelation of this relation, obvious candidates being subrelations of the form $\forall$–$\exists$–… of which there are two possibilities to consider, namely $\forall \mathit{Abs\text{-}Op} \exists \mathit{Conc\text{-}Op}(G \vee C)$ and $\forall \mathit{Conc\text{-}Op} \exists \mathit{Abs\text{-}Op}(G \vee C)$. We discuss these in turn.

If we take the $\forall \mathit{Abs\text{-}Op} \exists \mathit{Conc\text{-}Op}(G \vee C)$ form, we must consider four things. Firstly, this form makes the resulting proof obligation resemble a refinement from concrete to abstract systems (aside from P and C of course), taking us in a direction we do not intend to go. Secondly, the $\forall \mathit{Abs\text{-}Op}$ part forces us to say something about all possible abstract steps. There may be many of these that are quite irrelevant to the more definitive concrete system, since the abstract system is intended to be merely a simplifying guide to the concrete one; the necessity of mentioning them, or excluding them via the P clause, would be an unwelcome complication. Thirdly, this form does *not* make us say something about all possible concrete steps, limiting its usefulness as an enhancement to the description of the concrete system. And fourthly, we have not identified any negative criterion that we must ensure the abstract system fulfils, as was the case for concrete systems in refinement. All of these considerations mitigate against adopting the $\forall \mathit{Abs\text{-}Op} \exists \mathit{Conc\text{-}Op}(G \vee C)$ form.

So we turn to the $\forall \mathit{Conc\text{-}Op} \exists \mathit{Abs\text{-}Op}(G \vee C)$ form. Here we consider three points. Firstly, we are not required to say something about all abstract steps, which in view of the remarks above we regard as beneficial. Secondly, we must say something about all concrete steps, which helps to enhance the description of the concrete system, and which we thus regard as good. In particular we must consider for any concrete step, whether it is: (a), excluded from consideration because P is not validated; (b) included but requires essential use of C to satisfy the obligation; (c), included but does not require C . The third point follows on from (c): there may well be sensible portions of the state and I/O spaces in which P and C are trivial. In such places, when both trm clauses hold, it will be possible to derive from the truth of the retrenchment obligation, the truth of the refinement obligation; this would tie in neatly with the joint initialisation proof obligation $PA_{M,N} \Rightarrow [\ Y(v)\ ] \neg [\ X(u)\ ] \neg\ G(u, v)$ mentioned above. Such a state of affairs supports our intention that retrenchment is regarded as a liberalisation of refinement, i.e. it is like refinement "except round the edges".

The heuristic reasoning above aimed to justify a proof obligation that is simple and convenient to mechanise and to use in real designs; and in the light of the preceding remarks we see that saying that retrenchment is merely a "the strengthening of the precondition and the weakening of the postcondition" is deceptively simplistic. The current lack of a more abstract underlying model for retrenchment is not regarded as a fundamental obstacle to its usefulness, though such a model would clearly be of great interest. Indeed, as the third point above indicated, we can expect at minimum, various special cases of retrenchment to lend themselves to deeper mathematical treatment. It might be that there is no "best" such theory and that increasing ingenuity will reveal increasingly complex special cases. The inevitable consequence of this would be, that treated in a standalone fashion, the special cases would generate standalone proof obligations of an increasingly complex and thus less practically convenient nature. Such an outcome would strongly support our strategy of defining retrenchment directly via a simple proof obligation. These fascinating matters will be explored more fully elsewhere.

3.3 Composability of Retrenchments

We have deliberately designed retrenchment to be a very flexible relation between machines, to afford designers the maximum convenience and expressivity in constructing complex solutions to complex problems by reshaping oversimplified but more comprehensible pieces. We indicated above that the mathematics of retrenchment will be more complex than that of refinement and we do not embark on a full discussion here. Nevertheless we show here that retrenchments compose to give retrenchments. To see this suppose machine $N (b)$ is retrenched to machine $O (c)$ whose structure is defined by “schematically alphabetically incrementing” $N (b)$ as follows:

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	
$p \longleftarrow OpName (j) \triangleq$		$q \longleftarrow OpName (k) \triangleq$	
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	
$C (u , v , o , p , A)$		$D (v , w , p , q , B)$	
END		END	
END		END	

The initialisation proof obligation for N and O is then:

$$PA_{N,O} \Rightarrow [\ Z(w)\ ] \neg [\ Y(v)\ ] \neg H(v, w)$$

and the retrenchment proof obligation for N and O becomes:

$$\begin{aligned} PA_{N,O} \wedge (J(v) \wedge H(v, w) \wedge K(w)) \wedge (\mathsf{trm}(U(w, k, q)) \wedge Q(j, k, v, w, B)) \\ \Rightarrow \mathsf{trm}(T(v, j, p)) \wedge [\ U(w, k, q)\ ] \neg [\ T(v, j, p)\ ] \neg \\ (H(v, w) \vee D(v, w, p, q, B)) \end{aligned}$$

It is easiest to work in a relational framework. To this end suppose a takes values in the set A , u takes values in the set U , i takes values in the set I , o takes values in the set O , and so on, each variable we have mentioned taking values in a set named after its upper case sans serif analogue.

Now the initialisation proof obligation for N and O tells us that for each initialisation of O by $w \in \mathsf{W}$ there is an initialisation of N by $v \in \mathsf{V}$ such that $H(v, w)$ is true. And for this $v \in \mathsf{V}$ we know that $J(v)$ holds moreover by the initialisation proof obligation for N alone. Considering the initialisation proof obligation for M and N likewise tells us that for the given $v \in \mathsf{V}$ there is an initialisation of M by $u \in \mathsf{U}$ such that $G(u, v)$ holds. So we obtain:

$$PA_{M,N,O} \Rightarrow [\ Z(w)\ ] \neg [\ X(u)\ ] \neg (\exists\ v \bullet G(u, v) \wedge J(v) \wedge H(v, w))$$

In this, $(\exists\ v \bullet G(u, v) \wedge J(v) \wedge H(v, w))$ is revealed as the retrieve relation for the composition. For the retrenchment proof obligation for the composition we argue as follows. Let us assume for arbitrary u , i , w , k , q , B we can find v , j , A which witness:

$$\begin{aligned} PA_{M,N,O} \wedge (I(u) \wedge (\exists\ v \bullet G(u, v) \wedge J(v) \wedge H(v, w)) \wedge K(w)) \wedge \\ \big(\mathsf{trm}(U(w, k, q)) \wedge (\exists\ v\ , j\ , A \bullet G(u, v) \wedge J(v) \wedge H(v, w) \wedge \\ P(i, j, u, v, A) \wedge Q(j, k, v, w, B))\big) \end{aligned}$$

and let u_0 , i_0 , w_0 , k_0 , q_0 , B_0 , v_0 , j_0 , A_0 be an arbitrary choice of values that makes the unquantified version of this formula true. Since for this choice of values we have the antecedents of the retrenchment proof obligation for N and O , we conclude that $\mathsf{trm}(T(v_0, j_0, p_0))$ holds for suitable values p_0 . Likewise, since we now have the antecedents of the retrenchment proof obligation for M and N , we deduce:

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

for a suitable o_0 , which will be the first conjunct of a composed proof obligation.

The second conjunct of the composed obligation is derived along the following lines. Under our assumptions, the retrenchment proof obligation for N and O assures us that $[\ U(w, k, q)\ ] \neg [\ T(v, j, p)\ ] \neg (H(v, w) \vee D(v, w, p, q, B))$, i.e. that for each step of U there is a step of T such that $H(v_0{}', w_0{}') \vee D(v_0{}', w_0{}', p_0, q_0, B_0)$ is true, where the primes indicate the after state values and p_0 is chosen suitably.

Analogously, the retrenchment proof obligation for M and N assures us that $[\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg (G(u, v) \vee C(u, v, o, p, A))$, i.e. that for each step of T , and in particular for the one we have just chosen above, there is a step of S such that $G(u_0{}', v_0{}') \vee C(u_0{}', v_0{}', o_0, p_0, A_0)$ holds, where again o_0 is suitably chosen.

Putting these together, and noting that $J(v_0{}')$ holds anyway by machine consistency for N , we deduce that:

$$\begin{aligned} & J(v_0{}') \wedge \\ & \quad (G(u_0{}', v_0{}') \vee C(u_0{}', v_0{}', o_0, p_0, A_0)) \wedge \\ & \quad (H(v_0{}', w_0{}') \vee D(v_0{}', w_0{}', p_0, q_0, B_0)) \end{aligned}$$

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 dependent on them, allows us to write:

$$\begin{aligned} & PA_{M,N,O} \wedge (I(u) \wedge (\exists\ v \bullet G(u, v) \wedge J(v) \wedge H(v, w)) \wedge K(w)) \wedge \\ & \qquad \big(\mathsf{trm}(U(w, k, q)) \wedge (\exists\ v\ , j\ , A \bullet G(u, v) \wedge J(v) \wedge H(v, w) \wedge \\ & \qquad P(i, j, u, v, A) \wedge Q(j, k, v, w, B))\big) \\ & \quad \Rightarrow \mathsf{trm}(S(u, i, o)) \wedge [\ U(w, k, q)\ ] \neg [\ S(u, i, o)\ ] \neg \\ & \qquad \big(\exists\ v\ , p\ , A \bullet J(v) \wedge \\ & \qquad\quad (G(u, v) \vee C(u, v, o, p, A)) \wedge \\ & \qquad\quad (H(v, w) \vee D(v, w, p, q, B))\big) \end{aligned}$$

Using the distributive law, and weakening some of the derived disjuncts by discarding some $J(v)$ terms leads to:

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

This is now in the right shape for being interpreted as a retrenchment proof obligation for a composed retrenchment between M and O :

```
MACHINE          O ( c )
RETRENCHES       M
VARIABLES        w
INVARIANT        K ( w )
RETRIEVES        ∃ v • G ( u , v ) ∧ J( v ) ∧ H ( v , w )
INITIALISATION   Z ( w )
OPERATIONS
    q ⟵ OpName ( k ) ≙
        BEGIN
            U ( w , k , q )
        LVAR
            B
        WITHIN
            ( ∃ v , j , A • G ( u , v ) ∧ J( v ) ∧ H ( v , w ) ∧
                P ( i , j , u , v , A ) ∧ Q ( j , k , v , w , B ) )
        CONCEDES
            ( ∃ v , p • G ( u , v ) ∧ D ( v , w , p , q , B ) ) ∨
            ( ∃ v , p , A • C ( u , v , o , p , A ) ∧ H ( v , w ) ) ∨
            ( ∃ v , p , A • C ( u , v , o , p , A ) ∧ D ( v , w , p , q , B ) )
        END
END
```

We take this as the natural definition of composition of retrenchments, and we note that it is built out of the syntactic pieces of the component retrenchments in such a manner that composition of retrenchments will be associative.

We point out straight away that the above is not the only possible definition: an easy variation on what is given includes the $J(v)$ that we discarded in the three existentially quantified clauses of the CONCEDES clause of the composition. We dropped the $J(v)$ in all of them because designers are likely to be most interested in the interaction of the WITHIN and CONCEDES clauses in practical situations and the additional presence of $J(v)$ is likely to be seen only as a complicating nuisance.

We observe that the composed retrenchment is a different retrenchment than that obtained by alphabetically incrementing $N\,(\,b\,)$, even though the same machine $O\,(\,c\,)$ is involved. This is best understood from a categorical perspective. We could define a category $\mathcal{M}ch^{\mathcal{R}et}$ say, of machines and retrenchments, in which machines (given by the normal syntactic data for machines, and assuming (for convenience) that the machine name uniquely identifies the machine) constitute the objects, and abstract retrenchments (given by the syntactic data of the names of the retrenched and retrenching machines, the retrieve clause, and the ramification data) constitute the arrows. Roughly speaking, the law of composition of retrenchments, is the associative law of composition of arrows in this category[1]. The fact that there may be many different retrenchments to a given machine reflects the fact that there may be many different arrows to an object in a category, even many from the same source object. The concrete syntax proposed for retrenchments mixes object and arrow aspects in a single construct, and a good case could be made for their separation, especially since the connection between retrenched and retrenching machines is looser than in refinement. It is this categorical perspective that caused us to separate the local INVARIANT J from the RETRIEVE relation G in the definition of a retrenchment. And one could say much the same things about the usual refinement notion of course.

3.4 Simple Examples

With the preceding machinery in place, we can redo the earlier *My_Divine_Machine* / *My_Mundane_Machine* example properly as a retrenchment. We give first a minimal but self contained version of *My_Divine_Machine* :

```
MACHINE          My_Divine_Machine_0
VARIABLES        aa , bb , cc
INVARIANT        aa ∈ D-NAT ∧ bb ∈ D-NAT ∧ cc ∈ D-NAT
INITIALISATION   aa := 3 || bb := 4 || cc := 5
OPERATIONS
    MyPlus ≙ cc := aa + bb
END
```

And now we give a retrenchment of it along the lines of the original *My_Mundane_Machine* .

```
MACHINE          My_Mundane_Machine_1
RETRENCHES       My_Divine_Machine_0
VARIABLES        aaa , bbb , ccc
INVARIANT        aaa ∈ M-NAT ∧ bbb ∈ M-NAT ∧ ccc ∈ M-NAT
RETRIEVES        aa = aaa ∧ bb = bbb ∧ cc = ccc
INITIALISATION   aaa := 3 || bbb := 4 || ccc := 5
OPERATIONS
    resp ⟵ MyPlus ≙
        BEGIN
            IF
                aaa + bbb ≤ MaxNum
            THEN
                ccc := aaa + bbb ||
                resp := TRUE
            ELSE
                resp := FALSE
```

1. We say “roughly speaking”, because there are minor irritations concerning the identities and associativity in such a syntactic category since eg. $true \wedge true$ is semantically but not syntactically the same as $true$. One can circumvent these by defining the arrows as equivalence classes of syntactic data up to tautology (identifying eg. $\phi \wedge true$ with ϕ). We will not go into details.

```
            END
        LVAR
            CC
        WITHIN
            CC = ccc
        CONCEDES
            aa = aaa ∧ bb = bbb ∧ CC = ccc
        END
END
```

We are able to describe the case in which the variable *ccc* is not changed, with the help of the logical variable *CC* , noting that the CONCEDES clause refers to the after condition of the variables involved and that *CC* is not substituted. This situation is evidently outside the scope of normal refinement. To illustrate the flexibility of the retrenchment concept we give another, different retrenchment of *My_Divine_Machine* :

```
MACHINE           My_Mundane_Machine_2
RETRENCHES        My_Divine_Machine_0
VARIABLES         aaa
INVARIANT         aaa ∈ M-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
        WITHIN
            bb = bbb
        CONCEDES
            (resp = TRUE ⇒ cc = ccc) ∧ (resp = FALSE ⇒ ccc = 0)
        END
END
```

Note that in this version, the status of *bb* has been changed to that of an input and the status of *cc* has been changed to that of an output, thus obviating the need to take their properties into account in the RETRIEVES clause. The WITHIN and CONCEDES clauses instead take on the jobs of relating *bb* and *bbb* , and *cc* and *ccc* respectively; and in this simple example the RETRIEVES clause always holds anyway.

These two examples are rather trivial. Nevertheless they are small enough to show some of the technical details of retrenchment clearly. In Section 4 we will discuss two more convincing scenarios, albeit only superficially as a full treatment of either would require a substantial paper in its own right.

4 Continuous and Discrete Systems

In this section we discuss the prospects for applying retrenchment in the development of systems that need to model physical aspects of the real world, both in general terms and with regard to specific examples.

4.1 Modelling the Real World

As the application of formal methods for system development in the real world continues to grow, the interest in applying them to systems which capture the properties of physical situations requiring continuous mathematics for their description grows likewise. See for example [Maler (1997), Alur, Henzinger and Sontag (1996)]. In the bulk of such work the continuous component is time, and the problem is to describe and control a one dimensional dynamics typically governed by laws of the form

$$\dot{\boldsymbol{f}} = \Phi(\boldsymbol{f}, \boldsymbol{e})$$

where $\boldsymbol{f}$ is a (tuple of) quantities of interest, $\dot{\boldsymbol{f}}$ is the tuple of their first order time derivatives, and $\Phi(\boldsymbol{f}, \boldsymbol{e})$ is a tuple of formulae in the $\boldsymbol{f}$ and the external input $\boldsymbol{e}$. In addition the typically smooth evolution of the system according to the above law is punctuated from time to time with certain discrete events which interrupt the overall continuity of the system's behaviour. Over the years, a large amount of work has been directed at taming the difficulties that arise.

However there are also increasingly problems that involve applied mathematics of a different kind. A typical case in point which we will look at in more detail below is dose calculation in cancer radiotherapy [Johns and Cunningham (1976), Khan (1994), Cunningham (1989), Hounsell and Wilkinson (1994)]. Here the problem to be solved centres on the Boltzmann transport equation [Huang (1963)], a three dimensional nonlinear integro-differential equation that describes the electron (or X-ray) density. Not only is this not a typical one dimensional problem, but there are no known exact solutions or calculation techniques for this equation applicable to the kind of spatial configurations of interest in practice; solutions to practical examples rely on heuristic techniques whose efficacy is gauged by comparison with experiment. No formal technique is ever going to be able to "justify" the procedures undertaken, on the basis of primitive axioms, in the same way that simple calculations with natural numbers are justified on the basis of the Peano axioms in a modern theorem prover. But that is not to say that automated support is out of the question.

The fact that continuous mathematics has been done with rigour for a century or more is good evidence that what has been done there is formalisable. The paucity of published material in this area is more a question of logicians' and computer scientists' ignorance of and/or distaste for the subject than any issue of principle. In fact continuous mathematics has received some attention from the mechanical theorem proving community lately [Harrison (1996)]. The cited work shows that a formal approach to analysis is entirely feasible, but is a big job. The sheer breadth of applied mathematics that may be needed in applications makes it clear that simply formalising a large general purpose body of continuous mathematics that would serve as a foundation for "all" formal developments where such mathematics is required is an unrealistic proposition.

The alternative, to reinvent the core continuous mathematics wheel formally as a precursor to the more specialised reasoning required in a specific application is equally unrealistic. Not only would the cost of rederiving any significant piece of applied mathematics from first principles be prohibitive, but many different developments would overlap significantly in the mathematics needed despite the remarks above, and this would lead to wasteful duplication.

Moreover applied mathematics does not work by deriving everything from first principles. Rather, it reaches a certain stage of maturity, turns the results obtained into algebra, and uses the equations of that algebra as axioms for further work. Thus it seems reasonable for computerised developments that depend on such mathematics to select a suitable suite of already known results as axioms (whether these be formally derived or merely results that have achieved equivalent status through years of successful use), to capture these as axioms in the theory underlying the development, and to use these to support the specifics of the development. Where one starts from in the vast sea of extant mathematics to select an axiom basis which will be both useful in providing good support for the development at hand, and also tractable for automated reasoning technology, would become a matter for engineering judgement.

Even the heuristic semi-empirical reasoning alluded to above can be incorporated in such an approach. Suppose for example that it is believed that within certain bounds of applicability, such and such a parametrised expression can, by suitable choice of parameters, yield a function that is within such and such an error margin of a solution to a particular nonlinear integro-differential equation (say). Then that belief can be expressed as a rule in the system and its use controlled by the same theorem proving environment that supports the rest of the development.

At the moment, to the extent that developments incorporating the passage from continuous to discrete mathematics are attempted at all using a formal approach, what one sees is a little disconcerting. Typically some continuous mathematics appears, describing the problem as it is usually presented theoretically. When this is done, there comes a violent jolt. Suddenly one is in the world of discrete mathematics, and a whole new set of criteria come into play. There is almost never any examination of the conditions under which the discrete system provides an acceptable representation of the continuous one, and what "acceptability" means in the situation in question. But surely these questions are of vital importance if the discrete model is truly to be relied on. Results from mathematics which have investigated the reliability of discrete approximations to continuous situations have shown that there are useful general situations in which the discrete approximation can be depended on. Such results ought to make their way into the justification of the appropriate development steps in real world applications wherever possible. Evidently incorporating them into strict refinement steps is too much to ask in general, and the greater flexibility of the retrenchment formalism we propose seems to us to be much better suited to the task in hand.

4.2 A Furnace Example

We describe in outline a hypothetical situation in which the flexibility of retrenchment comes into its own. Suppose we have undertaken to control an ultraefficient furnace, into which pulverised solid fuel is injected along with preheated air. Magnetic fields help to keep the resulting burning ionised gas evenly suspended throughout the volume of the combustion chamber in order to maximise efficiency, and away from the walls in order to prevent them from disintegrating at the high temperatures used. The objective is to keep the temperature as high and as evenly distributed as possible for efficiency's sake, to keep the hot gas away from the walls, and to ensure that fluctuations do not cause explosive hot spots to arise that could give rise to shock waves which could damage the combustion chamber. A number of sensors in the combustion chamber report periodically on the temperature and electromagnetic flux in their vicinity. The physics of the situation is described by the partial differential equations of magnetohydrodynamics. Needless to say it is not possible to solve these equations in closed form. We want to model this situation in order to develop software that will control the magnetic fields and inflow of air and fuel, so as to maximise efficiency while keeping the system safe.

The top layer of the model simply reflects the classical mathematics of the problem. Thus suppose that the vessel occupies a volume Ω, with boundary $\partial\Omega$. The problem is then to control the flow of ionised gas, given by its temperature θ, mass density ρ, pressure p, adiabaticity γ, velocity $\boldsymbol{v}$, and current density $\boldsymbol{J}$, by adjusting the applied magnetic field $\boldsymbol{M}$, air input a, and fuel input s. The current values and rates of change of the physical quantities act as inputs, and the outputs are to be the future values of the physical variables and $\boldsymbol{M}$, a, s, such that over a finite ensuing period, the behaviour of the system is safe; i.e. no instabilities arise, and the ionised gas stays away from the solid parts of $\partial\Omega$. In B terms, one could have a single operation

$$\theta \ldots \boldsymbol{M} , a , s \longleftarrow \mathit{furnace\_control} (\theta , \rho , p , \gamma , \boldsymbol{v} , \boldsymbol{J} , \dot{\theta} , \dot{\rho} , \dot{p} , \dot{\gamma} , \dot{\boldsymbol{v}} , \dot{\boldsymbol{J}})$$

whose body was a relation which specified the future behaviour and the control outputs, against the input data. The substitution could be a typical ANY $\theta \ldots \boldsymbol{M}$, a, s WHERE *MHD_EQNs* END construct, where the body *MHD_EQNs* could simply quote the conjunction of the standard magnetohydrodynamic equations for the system with the desired bounds on future behaviour. This is specification at its most eloquent as the system is not capable of being solved in closed form, and the operation *furnace_control* would alter no state since we are at the textbook level of reasoning. Indeed the required relation could be captured in B constants, were it not for the desire to have an operation to retrench ultimately into an IMPLEMENTATION.

The intermediate layer of the model is a discretisation step in which continuous functions over space and time are replaced by finite sets of values at a grid of points in the combustion chamber Ω. So the operation would become something like

$$\theta_i \ldots \boldsymbol{M}_i , a_i , s_i \longleftarrow \mathit{furnace\_control} (\theta_i , \rho_i , p_i , \gamma_i , \boldsymbol{v}_i , \boldsymbol{J}_i , \dot{\theta}_i , \dot{\rho}_i , \dot{p}_i , \dot{\gamma}_i , \dot{\boldsymbol{v}}_i , \dot{\boldsymbol{J}}_i)$$

where the subscripts range over a suitable grid. Furthermore, while the top layer simply states the desired properties of the outputs, the intermediate layer now gives them as a more concrete function of the inputs, reflecting the structure of the finite element calculations needed to generate actual numerical answers, though not necessarily in full detail. The retrenchment between these layers would, if done thoroughly enough, cover the detailed justification for the discretisation. This would include bounds on the permitted fluctuations of the continuous system in order that it can still be adequately represented by the discrete system, as evidently not all violently fluctuating behaviours can be faithfully mirrored within a fixed grid. The mathematics required for the WITHIN and CONCEDES clauses, and to properly discharge the relevant proof obligations from first principles, would be demanding to say the least. It is likely that if this were a genuine development, some heuristic rules would be invoked to discharge the proof obligations, expressing accumulated engineering wisdom in such situations. The retrenchment would then be doing little more than documenting the arguments, but in a manner consistent with the rest of the development, and capable of some mechanical checking. There is still no state, so we still have pure I/O.

The lowest layer of the model takes the idealised finite element scenario above, and relates it to the actual configuration of input and output devices in the real system. Thus the only inputs at this level will be temperature and electromagnetic flux sensor readings, and the only outputs will be the control parameters to the air and fuel injectors and magnetic field controls. The signature will thus look like

$$\boldsymbol{M}_j \, , a_j \, , s_j \longleftarrow \textit{furnace\_control} \, (\, \theta_j \, , \boldsymbol{J}_j \,)$$

where j ranges over the actual input and output devices, and the remaining data of the intermediate model is committed to state variables of the machine. This arrangement is justified on the basis that the system changes sufficiently slowly that an iterative calculation of the required future behaviour can be done much faster and more accurately starting from the previous configuration, than ab initio from just the inputs. Furthermore, the model at this layer could stipulate numerical bounds on the values of the mathematical variables which occur, in order to ease the transition to computationally efficient arithmetic types later. The retrenchment from the layer above to this one will be rather easier than was the preceding retrenchment step, as the relationship between the I/O and state variables of the present model, and the I/O of the model above will consist of straightforward algebraic formulae, leading to relatively simple proof obligations. In particular the $\forall \textit{Conc-Op} \exists \textit{Abs-Op} (G \vee C)$ form of the retrenchment proof obligation enables the drawing up of a suitable C cognisant of these bounds rather more easily than the opposite form.

At this point we have reached the level that a conventional formal development might have started at. The operation of interest has reached a stage where its I/O signature and the information in its state has stabilised, so what remains is in the province of normal refinement. Such refinements could address the efficiency of the algorithms used, exploiting architectural features of the underlying hardware if appropriate, and could also address the precise representation of the state. We are assuming that the bounded mathematical types used in the model are such that casting them down to actual hardware arithmetic types can be done within a refinement; if not then another retrenchment would be required.

4.3 Radiotherapy Dose Calculations

We return now to radiotherapy dose calculations, to paint a similar picture in a perhaps more realistic setting. Radiotherapy is administered by shining a suitably collimated beam of either X-Rays or electrons at some part of the patient in order to irradiate a specific tumour volume inside the patient, the location of which is presumed known. The dose delivered is critical in that underdosing is just as harmful (in terms of not damaging the tumour sufficiently) as overdosing evidently is. In this problem there is no state as such, and operation calls merely represent the invocation of routines that calculate the output from a given set of input data. The conventional picture is that the geometrical data and beam intensity parameters are given, and the resulting dose is calculated; still we will return to this point later.

At the top level, the operation that calculates the delivered dose is described by simply stating the Boltzmann transport equation for the configuration of interest. As in the previous example, this is the essence of specification, stating the properties of the required radiation density implicitly, without any indication of how one might arrive at a solution.

As there is no exact means of solving the model, the next layer down attacks the problem by proposing a heuristic solution cast in term of a weighted sum of exponentially distributed basic radiation components. Such heuristic approaches, suitably parameterised, have been shown to give good agreement with experiment, i.e. to within a couple of percent of measured values, which is precise enough to fall within the accuracy requirements for radiotherapy. The retrenchment step between the two levels, and its proof obligations, expresses the faith that such an appropriately weighted sum of exponentials indeed yields something within acceptable error margins of the true solution. Note the flavour of the reasoning used here. The validity of the proposed approximation depends on the unstated assumption that the experimentally observed situation is indeed one that corresponds to a solution of the Boltzmann transport equation. This is a non-mathematical assumption. In principle it might well be the case that with enough effort expended on numerical analysis, one could independently establish that the heuristic did indeed approximately solve the equation, but in practice this is not done. In fact for radiotherapy, it is the physical world and the measurements it yields that provides the more important reference point for the development, so the Boltzmann transport equation might even be regarded as an irrelevance. However in practice, the physical insight into the world that the abstract transport equation model provides, is indispensible as a guide to pragmatic model building, even if the abstract model is not used directly.

A further retrenchment step can now take the heuristic solution, which is still very much in the continuous mathematics world, and turn it into a discretised model consisting of a finite collection of values on a suitable grid. The retrenchment proof obligations here may concern the smoothness assumptions of the continuous heuristic that allow the validity of the discrete approximation to be infered.

Yet another retrenchment step can now turn the above model into a specific numerical algorithm, perhaps expressed in terms of bounded numerical types, ready for implementation. And the stage is now set for a final step in which the algorithm is implemented, this last step being most probably a refinement if the preceding model was constructed with enough of an eye to the exigencies of the refinement proof obligations.

The above treatment of dose calculation is patterned on current practice in that one specifies the geometry and then one obtains the dose distribution, the rest happening outside the computational framework. But one can imagine an arrangement better suited to clinical reality, in which what one wants to specify is the desired dose distribution, and the required radiotherapy machine settings would be produced by the system. At the simplest level, this demands a reassignment of inputs and outputs in the model. However going further leads to greater difficulties, since in radiation theory terms, one is now dealing with an inverse problem rather than a direct problem. Nevertheless the prospects for mechanisation are not hopeless, since clinicians evidently solve these inverse models in some implicit fashion, otherwise they could never propose treatment regimes of any efficacy at all. Capturing the rules of thumb used in a formal way, essentially knowledge capture in the AI sense, would permit their mechanisation and incorporation into the computation framework described above. The resulting system would then look as follows.

The top level model would be much as before, except with inputs and outputs reassigned. Below, there would be an expert system trying to figure out machine settings to the best of its ability. Having chosen some settings, a dose calculation could be done more or less as described above, and the resulting dose distribution compared with the ideal one input, yielding one or more quality of fit indicators as additional output. If it was required to iterate this process over expert system "guesses", the dose calculation subsystem could be IMPORTed into the IMPLEMENTATION of the expert system, being called upon in successive iterations of the implementation's main loop.

4.4 Summary

Surveying the above two examples, we see how much of the reasoning that would otherwise fall outside of the remit of formal development can be brought into the fold by the use of retrenchment, at least in principle. Admittedly these examples were not worked out in full detail, but the outlines given show us how the engineer's model building activity may be organised within a formal process, and the ultimate very detailed and obscure model that is cast into implementation, may be made more approachable thereby. The whole development process also reveals the mathematically most challenging parts for what they are, and documents to what extent they have been resolved through utilising either deep results on discretisability from real analysis, or the adoption of pragmatic engineering intuition.

5 Conclusions

In the preceding sections we have argued that refinement is too restrictive to describe many developments fully, and have proposed retrenchment as a liberalisation of it. The objective was to allow more of the informal aspects of design to be formally captured and checked. We described the technical details of what retrenchment is within B, we considered some basic formal properties such as the proof obligations and composability, and discussed some examples.

Much of what we said regarding the applicability of retrenchment to realistic situations assumed the incorporation of ideal and richer types into B; this merits further discussion. Take the reals. Because the reals are based on non-constructive features, any finitistic approach to them will display weaknesses regarding what can be deduced, and different approaches will make different tradeoffs. (M-FLOAT is one possibility, and we could also mention different theories of constructible reals, as well as approaches that exploit laziness (in the functional programming sense) to yield so called computable exact reals.) The B attitude, to design a conservative framework for development, has the merit that a laudable degree of completeness of coverage can be achieved in the method. However to address many types of real world problem, this conservatism would need to be relaxed. It seems to us that the best way forward is to consider adding certified libraries to B, offering a variety of theories for richer types (eg. various types of reals), to give users the foundations for the applications they need. These idealisations could be retrenched away in the passage to an IMPLEMENTATION.

The lack of richer types in B is also felt at the I/O level, as B I/O only permits simple types to occur which thus can force premature concretisation. In this regard our work bears comparison with [Hayes and Sanders (1995)] who focus exclusively on I/O aspects, and who show how describing the I/O of an operation in excessively concrete terms, can lead to obscure specifications. Their decomposition of operations into an input abstraction phase, an abstract operation phase, and an output concretisation phase, corresponds to a special case of retrenchment in which there is no mixing of I/O and state aspects, but where the WITHIN and CONCEDES clauses permit translation from one I/O format to another. One can see this as further affirmation of the inadequacy of pure refinement as the only mechanism for turning abstract descriptions into concrete ones, as was indicated in Section 1.

The present work, the first on retrenchment, raises more questions than it solves. The true value of any development technique can only be judged by its usefulness in practice. For that, a significant body of case studies must be developed, and then those whose livelihood depends on doing developments right, must come to a verdict, either explicitly or implicitly, on whether the technique offers a worthwhile improvement on current practice or not. Retrenchment should be subjected to such critical appraisal in order to prove its worth — and how high in the abstraction ladder one starts a development process based on retrenchment is itself a matter for careful engineering judgement. Every formal process has its costs, even if only in terms of the labour needed to build a model within the stringent limitiations of a formal notation. These costs must be weighed against the potential benefits of greater integration of the overall process, and if mechanical checking and proof are done, the benefits derivable thereby. Finally, the authors envisage retrenchment as being useful both in continuous problems as discussed in the preceding section, and in entirely discrete situations too, where the complexity of the real system is built up in digestible steps from simpler models. We have given enough of the basic theory of retrenchment in this paper, to enable such application work and its evaluation to proceed. The other facet of retrenchment needing to be pursued, the underlying theory, aspects of which were discussed at the end of Section 3.2, is under active investigation and the results will be reported in future papers.

References

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

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

Alur R., Henzinger T., Sontag E. (eds.) (1996); Hybrid Systems III: Verification and Control. LNCS **1066**, Springer.

Back R. J. R. (1981); On Correct Refinement of Programs. J. Comp. Sys. Sci. **23**, 49-68.

Back R. J. R. (1988); A Calculus of Refinements for Program Derivations. Acta Inf. **25**, 593-624.

Back R. J. R., von Wright J. (1989); Refinement Calculus Part I: Sequential Nondeterministic Programs. *in*: Proc. REX Workshop, Stepwise Refinement of Distributed Systems, de Roever, Rozenberg (eds.), LNCS **430**, 42-66, Springer.

Bert D. (ed.) (1998); Proc. Second Int. B Conference. Recent Advances in the Development and Use of the B Method. LNCS **1393**, Springer.

Cunningham J. R. (1989); Development of Computer Algorithms for Radiation Treatment Planning. Int. J. Rad. Onc. Biol. Phys., **16**, 1367-1376.

Harrison J. R. (1996); Theorem Proving with the Real Numbers. PhD. Thesis, Cambridge University Computing Laboratory, *also* Cambridge University Computing Laboratory Technical Report No. 408.

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.

Hounsell A. R., Wilkinson J. M. (1994); Dose Calculations in Multi-Leaf Collimator Fields. *in*: Proc. XIth Int. Conf. on the use of Computers in Radiation Therapy, Manchester, UK.

Huang K. (1963); Statistical Mechanics. John Wiley.

Johns, H. E., Cunningham J. R. (1976); The Physics of Radiology. Charles C. Thomas.

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.

Khan F. M. (1994); The Physics of Radiation Therapy. Williams & Wilkins.

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

Maler O. (ed.) (1997); Hybrid and Real-Time Systems. LNCS **1201**, Springer.

Morgan C. (1994); Programming from Specifications. Prentice-Hall.

Morris J. M. (1987); A Theoretical Basis for Stepwise Refinement and the Programming Calculus. Sci. Comp. Prog. **9**, 287-306.

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

Wand I., Milner R. (eds.) (1996); Computing Tomorrow. Cambridge University Press.

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

von Wright J. (1994); The Lattice of Data Refinement. Acta Inf. **31**, 105-135.