Retrenchment: Extending Refinement for Continuous and Control Systems

M.R. Poppleton[†,‡] R.H. Banach[‡]

†Faculty of Mathematics and Computing,
Open University, Walton Hall, Milton Keynes MK7 6AL, UK
‡Department of Computer Science, Manchester University,
Manchester M13 9PL, UK
m.r.poppleton@open.ac.uk , banach@cs.man.ac.uk

June 30, 2000

Abstract

Discussion of a radiation dose calculation example demonstrates various expressive limitations of the refinement calculus, particularly for systems with continuous variables. A liberalization of refinement, called *retrenchment*, is proposed, which will support an analogous formal development calculus. Useful concrete system behaviour can be specified outside the domain of pure refinement, in particular behaviour under controlled precision decay.

A syntax and a formal definition are presented for retrenchment in the B notation of J.-R. Abrial. Necessary transitivity and monotonicity properties for a formal development calculus are stated. A generalisation, *evolving retrenchment*, is proposed, and a simple example demonstrates its utility, by analogy, in control systems applications. Evolution in retrenchment is demonstrated to offer the expressive power to describe useful simulation-like behaviour, with evolving precision, in software for control systems. Finally, the dosimetry problem demonstrates the architectural value of retrenchment for the formal construction of continuous systems.

1 Introduction

From early concerns about proving correctness of programs such as Hoare's [Hoa69] and Dijkstra's [Dij76], a mature *refinement calculus* of specifications to programs has developed via work such as [Bac81, Mor94, BvW89, HHS87]. The first relational proposal for a sound and complete proof method for data refinement was [HHS86]. A modern version of this *simulation* method, in its forward and backward forms, appears in [WD96, SCW98] in the Z notation [Spi93], and is discussed more generally in [dRE98].

In this context of model-based specifications the term "refinement" has a very precise meaning; according to Back and Butler [BB98] it is a "...correctness-preserving transformation...between (possibly abstract, non-executable) programs which is transitive, thus supporting stepwise refinement, and is monotonic with respect to program constructors, thus supporting piecewise refinement." Relationally, refinement is characterised as a development step requiring the concrete precondition to be weaker than the abstract (the applicability, or termination condition), and the concrete transition relation to be stronger, or less nondeterministic, than the abstract (the correctness, or transition condition). The most succinct characterisation of refinement is that of "operational indistinguishability", i.e. that every concrete behaviour be a possible abstract one.

Refinement is a strong technique in software development, both in descriptive power, and in delivering proof obligations that assert strongly coupled structure between levels of abstraction. So it is not

surprising that refinement cannot be used, without simplifying or approximating assumptions and informal justifications, in many real-world system situations. Early work on clean termination of programs [CH79, Bli81] shares our concern with refining specifications on abstract, infinite domains to finite computer-oriented domains. Another approach to this finiteness problem was Neilson's thesis [Nei90], which proposed a notion of acceptably inadequate design, i.e. that refinement over an infinite domain could be regarded as a limit of finite refinements. Partial logic approaches have also been proposed [Owe93].

This work is concerned with our proposal of *retrenchment*, a liberalisation of refinement, and its utility for formally verifiable software construction. We have argued, when first proposing the notion [BP98, PB99], for a weakening of the retrieve relation over the operation step, allowing concrete non-simulating behaviour in retrenchment. Concrete I/O may have different type to the abstract counterpart, and moreover the retrenchment relation may define fluidity between state and I/O components across the development step from abstract to concrete model. In this paper we recap the formal definition of retrenchment, and focus on questions of simulation (i.e. the existence of retrenchment, or refinement-like relationships between arbitrary operation sequences) and architecture. A motivating example in radiation dosimetry raises the issues, particularly that of precision decay in limited-precision and finite software.

The paper proceeds as follows: section 2 gives a brief resumé of the B notation of Jean-Reymond Abrial [Abr96], which is the formal framework for the presentation of the work. Section 3 presents a motivating example of dose calculation in radiotherapy [JC76, Kha94, HW94], leading to a formal definition of retrenchment. The dosimetry example is (partially) recast as a retrenchment. Section 4 states the transitivity and monotonicity results required for a full calculus of retrenchment. In section 5 the generalising notion of *evolving retrenchment* is proposed, with a motivating example. This example demonstrates that "evolution" of retrenchments provides the "glue" to provide useful simulation-like properties under retrenchment. By analogy, the utility of evolving retrenchment in control systems specification is discussed. Section 6 discusses the generalised monotonicity result for operation sequence over evolving retrenchment: this gives the formal basis for analysis of simulation-like behaviour in retrenchment. Section 7 concludes.

2 Résumé of B

B is based on a total correctness theory of programming. Its central construct is the predicate transformer, called a *generalised substitution*: $[S]R$ (more conventionally written $wp(S, R)$) describes the weakest precondition, or most general before-state from which program S is guaranteed to terminate satisfying postcondition R. In B, $[\cdots]$ distributes over conjunction and is monotonic w.r.t. implication. $[\cdots]$ does not satisfy Dijkstra's [Dij76] "Law of the Excluded Miracle" (which would require that $[S]\mathsf{false} \equiv \mathsf{false}$): this allows a notion of *feasibility* of programs. Programs (in general nondeterministic) are written in B using constructors inspired by Dijkstra's Guarded Command Language, called the Generalised Substitution Language (GSL). The basic operation is the *simple substitution* (which is assignment, in procedural programming terms). For replacement of free variable x in formula R by expression E (no free variable in E clashes with any bound variable in R) we write $[x := E]R$. The remaining simple constructors of B are axiomatised as follows (for unbounded choice z is nonfree in R; this will be written $z \setminus R$):

$[\mathit{skip}]R \equiv R$		skip
$[P \mid S]R \equiv P \wedge [S]R$		precondition
$[S [] T]R \equiv [S]R \wedge [T]R$		bounded choice
$[P \Longrightarrow S]R \equiv P \Rightarrow [S]R$		guard
$[@z \bullet S]R \equiv \forall z \bullet [S]R$	$z \setminus R$	unbounded choice

GSAxioms

The precondition constructor represents an explicit strengthening of the termination set, guard a strengthening of the feasibility set, bounded choice a demonic nondeterministic choice between two operations, and

unbounded choice a universally quantified demonic choice over all operations indexed on some (external) variable.

Any operation S working with a state variable (list) x can be expressed in the following *normalised* form, where P is a predicate in variable(s) x, Q is a predicate in variables x and x' (x' distinct from x):

$$S = P \mid @x' \bullet (Q \Longrightarrow x := x') \qquad \textbf{Norm}$$

From the axioms, this means that for any predicate $R(x)$

$$[S]R \equiv P \wedge \forall\, x' \bullet (Q \Rightarrow [x := x']R)$$

In fact, this decomposition into predicates P and Q is unique (modulo logical equivalence of predicates), and these are called $\mathsf{trm}(S)$ (termination predicate: before-states from which S is guaranteed to terminate) and $\mathsf{prd}_x(S)$ (before-after transition predicate) respectively. The latter form of this theorem gives a nice interpretation of S as a predicate transformer: from initial state x, S establishes R precisely when S terminates at x and every x' reachable from x under S satisfies R. These predicates can be explicitly defined:

$$\mathsf{trm}(S) \equiv [S]\,\mathrm{true}$$
$$\mathsf{prd}_x(S) \mathrel{\widehat{=}} \neg\, [S](x' \neq x)$$

A relational model is defined in the obvious way (where s is the set of which the state variable x is a member) for the precondition set, and the before-after relation for S :

$\mathsf{pre}(S) \mathrel{\widehat{=}}$	$\{x \mid x \in s \wedge \mathsf{trm}(S)\}$	precondition set
$\mathsf{rel}(S) \mathrel{\widehat{=}}$	$\{x, x' \mid (x, x') \in s \times s \wedge \mathsf{prd}_x(S)\}$	transition relation

The abstract syntax of the GSL is expressed in and complemented by the concrete syntax of the Abstract Machine Notation (AMN), which includes constructs for modular structuring. The unit of modularity is the *machine*, which contains inter alia a state *variable* (list), an *invariant* predicate expressing type and other required state constraints, an *initialisation*, and a set of *operations*, which are expressed in terms of state, input and output variables. The following syntax shows an abstract machine and a refinement. The latter is a derivative construct: its invariant clause $J(u, v)$ provides both local variable type and constraint information, as well as the retrieve relation to the abstract variable.

MACHINE	$M(a)$	REFINEMENT	N
		REFINES	M
VARIABLES	u	VARIABLES	v
INVARIANT	$I(u)$	INVARIANT	$J(u, v)$
INITIALISATION	$X(u)$	INITIALISATION	$Y(v)$
OPERATIONS		OPERATIONS	
$S(u, i, o) \mathrel{\widehat{=}} \cdots$		$T(v, i, o) \mathrel{\widehat{=}} \cdots$	
END		END	

RefSyn

A sufficient condition for refinement (equivalent to classical forward simulation) is expressed relationally as follows. Two abstract machines M and N are defined on state spaces u and v respectively, with a total relation (the *retrieve relation*) $r : v \leftrightarrow u$, and a bijection between the operations of M and N (say, every operation S of machine M corresponds to exactly one operation T of N). If for every such pair (S, T) the following hypotheses hold, then M is refined by N (written $M \sqsubseteq N$)[1]:

[1] A full B definition of refinement is omitted here; see [Abr96]

$v : c \leftrightarrow b \wedge$
$\mathrm{dom}(v) = c \wedge$
$v^{-1}[\mathsf{pre}(S)] \subseteq \mathsf{pre}[T] \wedge$
$v^{-1}; \mathsf{rel}(T) \subseteq \mathsf{rel}(S); v^{-1}$
$\Rightarrow$
$M \sqsubseteq N$ **RefR**

This corresponds to the operation refinement proof obligation (POB) in the following full formulation of the refinement rules in generalised substitution form (syntax as per RefSyn). The obligations are *initialisation consistency*, *operation consistency* (given abstract invariant and abstract operation termination, then the operation establishes the invariant), *initialisation refinement*, and *operation refinement* (for any concrete step of T, there is some abstract step of S that establishes the retrieve relation):

$[X]I$ **Init**
$I \wedge \mathsf{trm}(S) \Rightarrow [S]I$ **OpCons**
$[Y]\neg\,[X]\neg\, J$ **InitRef**
$I \wedge J \wedge \mathsf{trm}(S) \Rightarrow [T]\neg\,[S]\neg\, J$ **OpRef**

3 From refinement to retrenchment

Consider the specification of a program for dose calculation in radiotherapy. Concluding remarks will discuss the larger architectural picture of this highly nontrivial, safety-critical application and the relevance of retrenchment, but for now we specify a simple, empirically based program. In traditional radiotherapy, a beam of radiation (X-rays, electrons or heavy particles) was square, rectangular or circular in shape in intersection with the patient. Customised lead shielding was placed on the patient if any other beam intersection (field) shape was required, although today irregular fields can be shaped by collimator equipment at the beam head [HW94]. It is critical to neither underdose nor overdose the tumour, and to minimise damage to other healthy tissue by minimising healthy tissue volume inside the beam volume.

Practical dosimetry is empirically based, building tables of dose data collected by measuring exposure in "phantom" patients, i.e. dummies of the same shape and radiological characteristics as a human patient (usually tanks of water of size one cubic metre). These tables are usually based on circular beam fields, and are parameterised by field radius, beam quality, beam source to patient surface distance (SSD) and depth inside patient. Our example is a typical dosimetry problem of calculating (with acceptable accuracy) dose at depth d, with beam quality Q, SSD F, for a rectangular field of size $a * b$ cm^2.

These empirical approaches are based on various mathematical models, all of which respect the following basic radiation physics. Any particle (we include the photon as a "particle" here) of the radiation beam incident on the patient disperses its energy while traversing a path through the body. This path will be very irregular and characterised by many thousands of atomic interaction events that disperse the particle energy: an X-ray will attenuate and be deflected while dislodging electrons and releasing secondary Compton radiation from any atom in its path (very simply speaking - see [JC76] for the detailed physics involved). Thus, any small tissue volume in the beam volume can be regarded as subject to incident radiation of two kinds: *primary* radiation arriving directly from the beam source, and *scattered* radiation resulting from the myriad atomic scattering events.

Avoiding the physical details, we use the rad as unit of absorbed dose, which measures energy delivered per unit tissue mass. We use tables of the percentage-depth dose ratio $PDD(d, r, F, Q)$ and the backscatter

function $BS(r, Q)$. For field radius r, SSD F, beam quality Q, PDD represents the percentage of the surface dose delivered at depth d. For $r = 0$ the PDD represents the percentage of dose delivered purely from primary radiation. BS represents the ratio of dose at the patient surface, for radius r, to the dose in free space. Thus for $r = 0$, $BS = 1$: the backscatter measurement adds the dose "scattered back" from the patient through a beam field of radius r. For zero radius there is no scatter contribution to the backscatter ratio. Therefore, for radiation of 100 rads delivered in free air to the SSD for a beam field of radius r

$$\text{dose at depth } d = PDD(d, r, F, Q) * BS(r, Q) \tag{1}$$

For dose in free air other than 100 rads, simply multiply the dose at d accordingly. The scatter component of radiation at depth d, called the *scatter function*, is given by

$$S(d, r, F, Q) = PDD(d, r, F, Q) * BS(r, Q) - PDD(d, 0, F, Q) \tag{2}$$

The problem is less trivial for a field shape different from that tabulated, and the calculation is done by approximation and extrapolation. We wish to calculate the dose at point P, depth d, at the centre of the field size $a * b$, such as in figure 1. The method can be used to calculate dose at any point inside or outside the field.

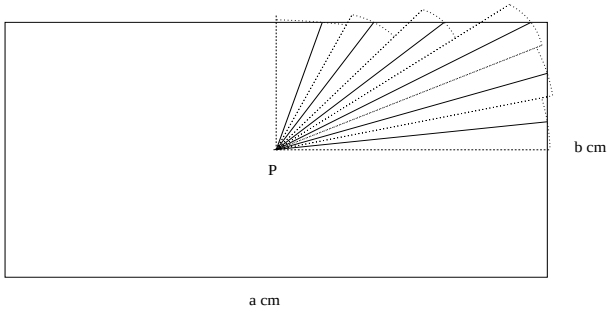


Figure 1: Circle-segment dose approximation for a rectangular field

We approximate the dose at P as the sum of the primary dose and of scatter components from circular segments of small angle and appropriate radius. Thus the dose at d for field $a * b$ is approximated by

$$\text{dose} = PDD(d, 0, F, Q) + \tfrac{1}{n} * \textstyle\sum_{i=1}^{n} S(d, r_i, F, Q) \tag{3}$$

Choosing segment angle 15^o as in figure 1 gives $n = 24$.

Now consider the specification of the dose calculation program and its refinement. For now we avoid the question of the empirical nature of the model described. We regard the abstract, ideal model of the physical world as one with real physical quantities described by BS_i and PDD_i, real-valued functions giving backscatter and percentage-depth-dose for rectangular fields. For simplicity we regard dose D_i, PDD_i and BS_i as state components. Thus in the abstract model the dose computation for depth d cm in a field of size $a * b$ cm^2 is simply (modifying signatures for rectangular fields)

$$D_i(d) = PDD_i(d, a, b, F, Q) * BS_i(a, b, Q) \tag{4}$$

Next consider how this might be refined to floating point (FP) arithmetic. Assume we have functions PDD_c and BS_c, which are FP counterparts to the ideal versions, still for rectangular fields. We need some retrieve relation, say defined in terms of limiting relative error on each of the state components (ideal i, concrete c, some given ϵ):

$$\mathsf{rel}(c,i) \mathrel{\widehat{=}} \ | \ \tfrac{c-i}{i} \ | \qquad (5)$$
$$\mathsf{ret}(c,i) \mathrel{\widehat{=}} \mathsf{rel}(c,i) \leq \epsilon$$

Immediately it is evident that the arithmetic precision loss makes refinement impossible. If components PDD_c, BS_c each have error less than ϵ, then in general D_c cannot - FP multiplication propagates error as follows:

$$\mathsf{rel}(D_c, D_i) = \mathsf{rel}(PDD_c, PDD_i) + \mathsf{rel}(BS_c, BS_i) - \mathsf{rel}(PDD_c, PDD_i) * \mathsf{rel}(BS_c, BS_i)$$

For argument relative errors close to zero, the result error increases. So refinement stops dead at this point; because there is no concrete step that guarantees exactly the same result as the abstract one (unless we restrict ourselves to a rather uninteresting fixed-precision subspace of the abstract model), the verification conditions generated by refinement are simply unavailable. This is cutting off the nose to spite the face; limited precision error propagation has been well understood for decades in the Numerical Analysis community (e.g. [Atk89]) and could be formally exploited in some more liberalised notion of the refinement step in software development. We will see that retrenchment addresses this need.

Having demonstrated the need to liberalise refinement, we will pursue the example to see the architectural utility of such a more liberal method. The approximate calculation based on segmented scatter functions as per (3) above is really a second development step. Assuming for now that there is some way to "liberally refine" the ideal model to the D_c, PDD_c, BS_c model, we now seek to construct a further refinement to a segmented model D_s. We do not discard PDD, BS; rather we note that the ideal and concrete versions were idealisations, physical functions that we believe exist, describing properties of rectangular fields. We treat the empirically measured PDD and BS tables for circular fields as input to the segmented model: there is no pretence that these relate directly to the idealised versions. We require only that D_s , specified by a computation such as (3), approximates, i.e. "liberally refines" each of D_c and D_i.

We model this dose calculation as two development steps since there are two quite different approximation processes happening: from ideal real arithmetic to fixed precision arithmetic, and then from ideal rectangular-field properties, to empirical circular-field approximations for rectangular-field properties. The second step error is trigonometric in nature and is determined by the relative error in the coverage of the rectangular field of area A by segments of total area A_{seg}. It is reasonable to assume (the trigonometry is left as an exercise!) that this error is proportional to the fineness of granularity of segmentation, i.e. that there is some k such that for an n-segment model

$$\mathsf{rel}(A_{seg}, A) \leq \tfrac{k}{n}$$

Of course any retrieve relation between D_s, D_c is complicated by the arithmetic of fixed-precision summation.

Thus a "liberalised refinement" development step (to be called a *retrenchment* step from now on) would be useful architecturally. If it provides a language for describing the looser relationship between levels of abstraction, then it may allow us to separate distinct design decisions in the manner demonstrated. Of course, in any realistic development, once a model is produced "sufficiently close" to the finite, deterministic implementation platform, then "vanilla" refinement can be applied.

The following syntax attempts a refinement for the first development step discussed above; later it will serve as a starting point to specify the retrenchment. We pretend that B contains a richer language of types, à la Z or VDM, than it actually does: we will use REAL (reals as conventionally defined) as well as its mundane, finite counterpart FLOAT.

MACHINE *DivineMult* REFINEMENT *MundaneMult*

		REFINES	*DivineMult*
VARIABLES	di, pi, bi	VARIABLES	dc, pc, bc
INVARIANT		INVARIANT	
$di \in$ REAL		$dc \in$ FLOAT $\wedge$ $pc \in$ FLOAT $\wedge$ $bc \in$ FLOAT	
$\wedge$ $pi \in$ REAL		$\wedge$ ret(dc, di) $\wedge$ ret(pc, pi) $\wedge$ ret(bc, bi)	
$\wedge$ $bi \in$ REAL			
OPERATIONS		OPERATIONS	
$DMult \mathrel{\widehat{=}}$		$resp \longleftarrow MMult \mathrel{\widehat{=}}$	
$di := pi * bi$		IF notOf(pc, bc) THEN $dc := pc * bc$ ELSE skip END	
$\cdots$		$\|\|$ $(resp := T \; [] resp := F)$	
END		$\cdots$	
		END	

It suffices briefly to reflect on the ways in which this is not a refinement. The retrieve relation (i.e. INVARIANT clause in the REFINEMENT) is not total; overflowing state elements are not modelled. The operation signature has changed: the concrete operation returns a success/ fail response, since we cannot guarantee that relative error remains within ϵ, i.e. that ret(dc, di) is true. It seems useful to be able to add such structure to the concrete model. There are two situations in which an abstract behaviour cannot be represented: if guard predicate notOf determines that the product will overflow, or if the result dc relative error exceeds ϵ. Hence *MMult* nondeterministically returns a T/F response, and resolution of this decision is left as a design decision for a subsequent step.

3.1 A syntax and a definition for retrenchment

To anticipate its definition, we propose that retrenchment be, very loosely speaking, the strengthening of the precondition and the weakening of the postcondition. This seems superficially like the opposite of refinement. However, we propose to liberalise the connection between abstract and concrete operations even more widely than that. We will not only allow changes of data type in the state component of an operation, but 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 designed 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.

We propose the following syntax for retrenchment. The square brackets indicate optional text.

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	
$o \longleftarrow OpName(i) \mathrel{\widehat{=}}$		$p \longleftarrow OpNameC(j) \mathrel{\widehat{=}}$	

```
        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
```

Abstract machine M has parameter a, state variable u, and invariant predicate $I(u)$. Variable u is initialised by substitution $X(u)$, and is operated on by operation *OpName*, a syntactic wrapper for substitution $S(u, i, o)$, with input i and output o. Unlike a refinement, which in B is a construct derived from the refined machine, a retrenchment is an independent MACHINE. Thus N is a machine with parameter b (not necessarily related to a), state variable v, invariant $J(v)$, initialisation $Y(v)$, and operation *OpNameC* as wrapper for $T(v, j, p)$, a substitution with input j and output p. Viewed as an independent machine, N cannot refer to M. So the new retrenchment syntax proposed here must in this case be regarded as null text.

Here we state that machine N RETRENCHES machine M; in general either a MACHINE or a REFINEMENT may be retrenched. The RETRENCHES clause (similarly to REFINES) makes visible the contents of the retrenched construct. We further assume that the name spaces of the retrenched and retrenching constructs are disjoint, but admit an injection of (retrenched to retrenching) operation names. An *injection* allows further, independent dynamic structure in the retrenching machine.

In the retrenching machine N we have state variable v under invariant $J(v)$, and the RETRIEVES clause $G(u, v)$. The existence of N as an independent machine requires that its (local) invariant be stated independently of the retrieve relation, unlike in B, where the two take the form of a joint predicate.

The radiation dosimetry example, in describing precision decay, showed it was desirable to express what an operation could achieve if it was not able to maintain an invariant retrieve relation. To give such expressiveness we propose that the relationship between concrete and abstract state is fundamentally different *before* and *after* the operation. We model this by distinguishing between a strengthened before-relation between abstract and concrete states, and a weakened after-relation. Thus the syntax of the concrete operation *OpNameC* in N is precisely as in B, with the addition of the *ramification*, a syntactic enclosure of the operation. We call this augmented syntax for the operation a *ramified generalised substitution*. Before, the relationship is constrained (precondition is strengthened) by the new *WITHIN* condition $P(u, v, i, j, A)$ which may change the balance of components between input and state, and may further define an optional "memory" variable LVAR A for reference in the after-state. The after-relation is weakened (postcondition is weakened) by the CONCEDES clause (the *concession*) $C(u, v, o, p, A)$, which specifies what the operation guarantees to achieve (in terms of after-state, output and logical variable A) if it cannot maintain the retrieve relation G. The concession will describe the weaker state of representation after the operation, or preserve exception information if representation fails completely.

We regard the proof obligations as central in the new theory; these give a semantic definition for retrenchment. The initialisation POB will be as for refinement, guaranteeing a joint starting state satisfying $G(u, v)$.

The following POB serves as the definition of retrenchment - the proof obligation (analogous to that for operation refinement) that concrete *OpNameC* (substitution T) retrenches the abstract operation *OpName* (substitution S):

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

RetGS

As shorthand we write $S \lesssim_{G,P,C,A} T$. We will justify this definition by comparison with the refinement proof obligation. In refinement, "operational indistinguishability" of the refined operation is the central issue. It follows that when the abstract operation terminates, so does the concrete one: precondition weakening. In retrenchment we are inverting this relationship. The retrenching machine is different from, independent of, and in a sense, bigger than (contains more information and structure) than the retrenched machine. It can simulate the abstract machine only for some before-states. Thus we may not be able to conclude anything from abstract operation termination; however, if we know the concrete operation terminates then we conclude that the abstract one must too. Thus we *strengthen* the precondition in retrenchment. We further strengthen this precondition by conjoining in the proof obligation hypotheses the WITHIN clause P. This further restricts the combinations of before-state and inputs (over and above those specified by the RETRIEVES clause) where the concrete operation (viewed as a retrenchment) is meaningful and will terminate. We have thus accounted for the inversion of the trm clauses, and the inclusion of the P clause in the hypotheses.

We next observe that the dynamic behaviour of the operation is guaranteed to achieve $(G \vee C)$, that is, either the RETRIEVES clause or the CONCEDES clause. The concession is defined in terms of abstract and concrete after-state and output, as well as the logical "memory" variable A. We also observe that we have preserved the familiar shape of the consequent, i.e. $\forall\, ConcOp \exists\, AbsOp \bullet (G \vee C)$. That is, for *every* concrete transition there exists *some* abstract transition that *either* achieves the retrieve relation *or* at least meets the concession. There are a number of benefits to this shape:

- We do not make a statement about *all* abstract steps; this is desirable and consistent with the need to reduce nondeterminism in retrenchment.
- We make a statement about *every* concrete step; this enhances the quality of the concrete description.
- Furthermore, for every concrete step, we must consider whether it is (i) excluded from consideration because either P or $\mathsf{trm}(T)$ are not satisfied, (ii) included but can only achieve the concession C, or (iii) included and achieves G. In fact in the latter case we have refinement-like behaviour, supporting our intuition that retrenchment is merely "refinement blurred around the edges"; this point will be further developed in due course.

We complete this section by expressing the concrete subtraction operation *MMult* as a retrenchment:

MACHINE	*MundaneMult*1
RETRENCHES	*DivineMult*
VARIABLES	dc, pc, bc
INVARIANT	$dc \in \mathrm{FLOAT} \wedge pc \in \mathrm{FLOAT} \wedge bc \in \mathrm{FLOAT}$
RETRIEVES	$\mathsf{ret}(dc, di) \wedge \mathsf{ret}(pc, pi) \wedge \mathsf{ret}(bc, bi)$

$resp \longleftarrow MMult \mathrel{\widehat{=}}$

$dc := pc * bc \;||\; (resp := T \;[]\; resp := F)$

WITHIN $\quad \mathsf{notOf}(pc, bc)$

CONCEDES $\quad resp := T \Rightarrow \mathsf{ret}(dc, di)$

$\quad \wedge\; resp := F \Rightarrow \mathsf{rel}(dc, di) \leq 2 * \epsilon$

...

END

The design decisions required before are now expressible in the retrenchment step. The IF-structure for *MMult* with the skip option is unnecessary since the retrenchment is only defined within the WITHIN clause $\mathsf{notOf}(pc, bc)$ as required. The concession is strong and informative here, in logically overlapping with the RETRIEVES clause: it tells us that the response output is the guaranteed indicator either of refinement ($resp := T$) or of controlled precision loss ($resp := F$). Nonetheless at this level of abstraction this choice of response remains nondeterministic, and thus a subsequent design decision.

4 A retrenchment calculus

As mentioned in the introduction, in order to make a *retrenchment calculus* analogous to the refinement calculus, it is necessary to show retrenchment is transitive, and that the constructors of the specification notation (here, the GSL of B) are monotonic w.r.t. retrenchment. For brevity we simply state the results here.

For transitivity, we assume (as before in section 3.1) that machine N RETRENCHES M, and further that machine O RETRENCHES N. We define machine O syntactically as a "lexicographic increment" on N, schematically replacing occurrences of $N,b,M,v,J,G,Y,\ p,j,T,A,P,C$ in N by $O,c,N,w,K,H,Z,q,k,U,B,Q,D$, respectively. Thus operation S in machine M is retrenched by operation T in machine N (w.r.t. G, P, C, A), which is in turn retrenched by operation U in machine O (w.r.t. H, Q, D, B).

Theorem RetComp

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

The result is intuitively satisfying. The RETRIEVES clause $\exists\, v \bullet (G \wedge J \wedge H)$ combines component RETRIEVES clauses and intermediate invariant. The WITHIN clause $\exists\, v, j, A \bullet (G \wedge \cdots \wedge Q)$ combines all component before-state RETRIEVES and WITHIN constraints. The concession is a cross-product of component RETRIEVES clauses and concessions. It can be shown that the transitivity property associates.

There are two monotonicity theorems; we leave the theorem concerned with operation sequence ";" for later. To avoid repetition the sequence result is given in its generalised "evolving" form in section 6.

Theorem RetMono : Retrenchment is monotonic w.r.t. the precondition, guard, bounded and unbounded choice and parallel constructors of B.

Case $\mid$ **:** Assume $S(u,i,o) \lesssim_{G,P,C,A} T(v,j,p)$. Assume Q is a predicate either on a common input variable or some shared external variable unaffected by either operation. Provided u, v, o, p are nonfree in Q, then

$$(Q \mid S) \lesssim_{G,P,C,A} (Q \mid T)$$

Case $\Longrightarrow$ **:** Assume $S(u,i,o) \lesssim_{G,P,C,A} T(v,j,p)$. Assume Q as for case $\mid$ above. Then

$$(Q \Longrightarrow S) \lesssim_{G,P,C,A} (Q \Longrightarrow T)$$

Case [] **:** Assume $S1(u,i,o) \lesssim_{G,P_1,C_1,A_1} T1(v,j,p)$ and $S2(u,i,o) \lesssim_{G,P_2,C_2,A_2} T2(v,j,p)$. Then

$$S1 [] S2 \lesssim_{G,P_1 \wedge P_2, C_1 \vee C_2, (A_1,A_2)} T1 [] T2$$

Case @ **:** Now assume that $S(u,i,o,x) \lesssim_{G,P,C,A} T(v,j,p,x)$ for any value of some fresh free external variable x (distinct from u, i, v, j, p, A). Then

$$@x \bullet S \lesssim_{G,P,C,A} @x \bullet T$$

Case $\mid\mid$ **:** Assume four operations in four distinct machines:

$$S1(u_1,i_1,o_1) \lesssim_{G_1,P_1,C_1,A_1} T1(v_1,j_1,p_1) \text{ subject to } I_1, J_1 \text{ respectively}$$
$$S2(u_2,i_2,o_2) \lesssim_{G_2,P_2,C_2,A_2} T2(v_2,j_2,p_2) \text{ subject to } I_2, J_2 \text{ respectively}$$

where all variables, inputs and outputs are distinct. Then

$$S1 \mid\mid S2 \lesssim_{G_1 \wedge G_2, P_1 \wedge P_2, C_{12}, (A_1,A_2)} T1 \mid\mid T2 \text{ subject to } I_1 \wedge I_2, J_1 \wedge J_2$$
... where
$$C_{12} \mathrel{\widehat{=}} (G_1 \wedge C_2) \vee (G_2 \wedge C_1) \vee (C_1 \wedge C_2)$$

5 Evolving retrenchment and continuous systems

To improve expressiveness in specification with retrenchment, the notion needs further specialisation. Being a very weak relation, retrenchment lends itself to application-domain specific strengthening. We have introduced various such specialisations (modulated refinement, sharp and simple simulable retrenchment - [BP99a, BP99b, BP00]) and here propose a notion of *evolving retrenchment* for use in simulation of continuous systems.

The notion of an evolving relationship between system models emerges from the intrinsically approximate nature of modelling real-world systems with continuous variables. For example, to model the motion of a projectile near to the earth we might use a linear differential equation relating the acceleration of the projectile to the forces of gravity and air friction. We might choose to omit, for practical reasons, the Coriolis effect, or the effect of time-varying crosswinds. So the best mathematical model available to us will always be approximate, for reasons of cost, practicality, or even lack of mathematical sophistication.

This work is, however, concerned with the relationship between mathematical models at adjacent levels of abstraction. In the projectile example we might (unwisely choosing a large reduction in abstraction level) choose to specify the next layer of model using some finite set of floating-point numbers appropriate to computer implementation. Clearly, a second level of approximation is introduced: before any system

dynamics are considered, precision is limited to that determined by the floating-point set chosen. Moreover, when some abstract (continuous) operation is considered in relation to its concrete (floating point) counterpart, precision of representation is in general reduced. For simple addition, precision decays by a factor of two. For a more complex operation and its "refinement", say exponentiation, the precision decay is far more complex.

From a refinement point of view this suggests that relaxing the invariant nature of the retrieve relation between abstract and concrete models, allowing it to *evolve*, might give a far more expressive development step than either refinement or "vanilla" retrenchment. Although retrenchment as discussed thus far allows breach of the invariant (via establishment of the concession), such breach effectively stops any simulation of the abstract operation sequence by the concrete. Specialisations of retrenchment, as discussed above, seek to combine the concession achieved with application domain structure to re-establish the retrieve relation where the simulation has failed to establish it. The approach here is more universal in relaxing the immutability of the retrieve relation, and allowing it to evolve. That is, if one operation step results under retrenchment in achievement of a weaker, evolved retrieve relation, this latter relation can serve as a starting point for a subsequent retrenchment step.

More formally we propose that the retrieve relation G becomes "variant" (i.e. evolves), in the sense that it becomes mediated by some parameter α. The intuition behind this is that α belongs to some ordered set, where increasing α denotes decreasing precision of the representation of abstract u by concrete v in $G_\alpha(u, v)$. That is, we will usually (but not always) expect that

$$\alpha \leq \beta \Rrightarrow (G_\alpha \Rightarrow G_\beta)$$

where "$\Rrightarrow$" reads "implies in all valuations". This formulation describes a typical precision-decay situation over a simulation, for example in a sequence of arithmetic steps (real - floating point). We choose the notion of *evolving* rather than *decaying* retrenchment as this suggests, because it is quite possible for precision to *increase* over a simulation step. In a control system, sensor readings provide the software with the inputs necessary to model the real system state. In control systems with a large number of sensors, it is quite likely that, at a given point in time, some sensors have failed, and that the modelled state is representing the real system state in some degraded mode[2]. Some algorithm extrapolating the failed sensor value from neighbouring sensors may be applied, to make the best of things, but accuracy is nonetheless lost. Assuming it is possible for sensors to be repaired in-flight, and for repair status to be detectable, it is manifestly possible for precision of the system state representation to improve.

Simple retrenchment is therefore generalised to define *evolving retrenchment*:

$$\begin{array}{l} I(u) \wedge G_\alpha(u, v) \wedge J(v) \wedge P_\alpha(i, j, u, v, A) \wedge \mathsf{trm}(T)(v, j) \\ \Rightarrow \mathsf{trm}(S)(u, i) \wedge [T(v, j, p)] \neg\, [S(u, i, o)] \neg\, (G_\beta(u, v) \vee C_\beta(u, v, o, p, A)) \end{array}$$

RetE

The following shorthand form will be more convenient:

$$S \lesssim T \quad \text{w.r.t.} \quad (G_\alpha, P_\alpha \longrightarrow G_\beta, C_\beta)$$

Thus the (u, v) relationship is now mediated through evolving precision parameters α and β. Notice that WITHIN and CONCEDES predicates are also parameterised: this is necessary since each of these further constrain the (u, v) relationship, in context of inputs and outputs respectively. Generalising these predicates also allows these relations between abstract/concrete I/O and state to evolve. Note that the

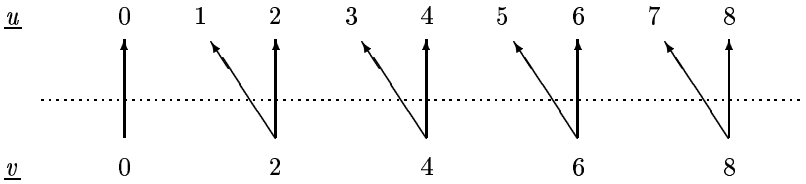
[2]This relies on reliable sensor failure determination, an issue we will not pursue here.

precision parameters are metavariables, parameterising the overall retrenchment relationship between abstract and concrete models, and are conceptually quite different from state or I/O variables. They denote the (usually decaying) evolution of the RETRIEVES relationships involved at each step.

A simple example in natural number arithmetic will demonstrate the use of evolving retrenchment and, by analogy, show its utility for floating-point arithmetic and for control systems applications. As before we take the ideal arithmetic of the naturals as abstract model. We assume loss of capability to represent odd numbers as the limitation of the concrete model: this is the even naturals and their approximation of the ideal natural arithmetic.

Figure 2: Evolving retrenchment of simple natural arithmetic

Abstract model: naturals



Concrete model: even naturals

We assume abstract inputs are natural, concrete inputs are even natural approximations of their abstract counterparts. Thus

$\underline{i} = \underline{u} =$ Abstract type $= \mathbb{N}$ $\quad +_a \mathrel{\widehat{=}} +_{\mathbb{N}} \quad -_a \mathrel{\widehat{=}} -_{\mathbb{N}}$

$\underline{j} = \underline{v} =$ Concrete type $= 2\mathbb{N}$ $\quad +_c \mathrel{\widehat{=}} +_{2\mathbb{N}} \quad -_c \mathrel{\widehat{=}} -_{2\mathbb{N}}$

where $2\mathbb{N} \mathrel{\widehat{=}} \{2 * n \mid n \in \mathbb{N}\}$

More precisely define

$+_a(u,i) \mathrel{\widehat{=}} u := u + i \qquad -_a(u,i) \mathrel{\widehat{=}} u \geq i \mid u := u - i$

$+_c(v,j) \mathrel{\widehat{=}} v := v + j \qquad -_c(v,j) \mathrel{\widehat{=}} v \geq j \mid v := v - j$

$\underline{i}$ and $\underline{j}$ are abstract and concrete input types. We have

$I(u) \mathrel{\widehat{=}} u \in \mathbb{N} \qquad P(i,j) \mathrel{\widehat{=}} 0 \leq j - i \leq 1 \wedge i \in \mathbb{N} \wedge j \in 2\mathbb{N}$

$J(v) \mathrel{\widehat{=}} v \in 2\mathbb{N}$

and the initial retrieve relation G is

$G(u,v) \mathrel{\widehat{=}} 0 \leq v - u \leq 1$

Both addition operations terminate vacuously; for subtraction we have

$\mathsf{trm}(-_a) \equiv u \geq i \qquad \mathsf{trm}(-_c) \equiv v \geq j$

The concrete model is thus one of limited precision representing even numbers exactly, and odd numbers as the next even number up. The input/output representation is the same; we choose "one-sided" representation rather than $\mid v - u \mid \leq 1$ to avoid repetitive proliferation of predicates that add nothing to the discussion.

Evidently, precision may decay under addition:

$3^u + 4^i \longrightarrow 7^{u'}$ $\qquad$ $3^u + 3^i \longrightarrow 6^{u'}$
$4^v + 4^j \longrightarrow 8^{v'}$ $\qquad$ $4^v + 4^j \longrightarrow 8^{v'}$
$v - u = 1$ $\qquad$ $v - u = 1$
$v' - u' = 1$ $\qquad$ $v' - u' = 2$

Therefore define

$$G_n(u, v) \mathrel{\widehat{=}} 0 \leq v - u \leq n \wedge n \in \mathbb{N}_1$$

It is trivial to see that we have a retrenchment where the weaker-precision G_{n+1} is the concession:

$$I \wedge G_n \wedge J \wedge P \Rightarrow [+_c]\neg\, [+_a]\neg\, (G_n \vee G_{n+1}) \tag{1}$$

since, given u, i, v, j and $P \wedge G_n$, for any $v' = v + j$ from a $+_c$-step, then for $u' = u + i$

$v' - u' = v + j - (u + i) = (v - u) + (j - i) \leq n + 1$
and $\quad v' - u' \leq n \quad$ for $\quad j = i$
Also, since $\quad v - u \geq 0 \wedge j - i \geq 0$
we have $\quad v' - u' \geq 0$

If we strengthen the WITHIN clause to force $j = i$, i.e. restrict abstract input to even numbers, we have a local refinement (in the sense of section 3):

$$P'_+(i, j) \mathrel{\widehat{=}} P(i, j) \wedge i = j$$
$$I \wedge G_n \wedge J \wedge P'_+ \Rightarrow [+_c]\neg\, [+_a]\neg\, G_n \tag{2}$$

For subtraction the WITHIN clause is a little more intricate: define

$$P_{-n}(i, j, u, v) \mathrel{\widehat{=}} P \wedge P_{can} \wedge P_{trm}$$
...where
$$P(i, j) \mathrel{\widehat{=}} 0 \leq j - i \leq 1 \wedge i \in \mathbb{N} \wedge j \in \mathbb{N}_1$$
$$P_{can}(i, j, u, v) \mathrel{\widehat{=}} v = u \Rightarrow j = i$$
$$P_{trm}(j, v, n) \mathrel{\widehat{=}} v - j \geq n$$

P gives the basic I/O relation as before. P_{can} avoids "over-cancellation of error" when $u = v$ and $i < j$, such as

$6^u - 1^i \longrightarrow 5^{u'}$
$6^v - 2^j \longrightarrow 4^{v'}$

Thus a limitation of the model is that when $u = v$, only subtraction of even inputs can be represented. P_{trm} is required in the proof of termination in the subtraction retrenchment:

$$I \wedge G_n \wedge J \wedge \mathsf{trm}(-_c) \wedge P_{-n}$$
$$\Rightarrow \mathsf{trm}(-_a) \wedge [-_c]\neg\, [-_a]\neg\, (G_n \vee G_{n-1}) \tag{3}$$

To prove termination: from P_{trm} infer $v - n \geq j$. From G_n infer $v - n \leq u$. Thus $u \geq v - n \geq j$. From P we have $j \geq i$, thus we have $u \geq i \equiv \mathsf{trm}(-_a)$.

To prove transition: given u, i, v, j, we have $u' = u - i$ and $v' = v - j$. Then

$$v' - u' = (v - j) - (u - i) = (v - u) - (j - i) \geq 0 \quad \text{given} \quad P_{can}$$

From that expansion we also have, from P and G_n that $v' - u' \leq n$ and, for $j - i = 1$, that $v' - u' \leq n - 1$. This proves (3) and also the stronger result

$$P'_{-n}(u,i,v,j) \mathrel{\widehat{=}} P_{-n} \wedge v \neq u \Rightarrow j - i = 1$$
$$I \wedge G_n \wedge J \wedge \mathsf{trm}(-_c) \wedge P'_{-n}$$
$$\Rightarrow \mathsf{trm}(-_a) \wedge [-_c]\neg\, [-_a]\neg\, G_{n-1} \qquad (4)$$

For addition we guarantee $+_a \lesssim +_c$ w.r.t. $(G_n, P \longrightarrow G_{n+1}, \mathsf{false})$, i.e. precision decay of at most one per operation step. There is a more optimistic reading of this, i.e. $(G_n, P \longrightarrow G_{n+1}, G_n)$: precision decay of at most one, with the possibility of precision maintenance. In a control system the first reading is analogous to loss of a sensor and a move to degraded mode with less accurate state representation. The optimistic reading (which might be less than optimistic for the control system!) corresponds to uncertainty as to whether a sensor has failed or not over the operation step.

For the stronger, localised case of even abstract inputs, we guarantee precision maintenance, i.e. refinement: $(G_n, P'_+ \longrightarrow G_n, \mathsf{false})$. This is analogous to refinement in the control system, maintaining accuracy with all sensors present before the operation step known to remain intact.

Similarly, subtraction under P_{-n} is analogous to precision maintenance in the control system: $(G_n, P_{-n} \longrightarrow G_n, \mathsf{false})$. A more optimistic reading allows the possibility of precision improvement: $(G_n, P_{-n} \longrightarrow G_n, G_{n-1})$. Again this corresponds in the control system to the situation where all sensors present before the operation step remain intact, with uncertainty as to whether a sensor has recovered during the step.

The stronger case P'_{-n} guarantees improvement in accuracy of one: $(G_n, P'_{-n} \longrightarrow G_{n-1}, \mathsf{false})$. That is, precision has improved with the detected recovery of a sensor.

The analogy we make between simple arithmetic and control system is crude but illustrative. Evolving retrenchment can describe the evolving relationship (and its accuracy) between modelled and real system state. It can moreover describe the uncertainty implicit in such real-world systems, due to such things as the stochastic nature of failure detection processes.

6 Sequence of evolving retrenchments

It appears that the generalisation to evolving retrenchment cascades straightforwardly through the construction of a calculus of evolving retrenchment analogous to that of section 4, modulo some rather baroque expressions composing the component precision parameters.

Recall the *simulation* property for refinement: given a collection of operation refinements, it follows that an arbitrary abstract operation sequence is refined by the corresponding concrete operation sequence. This follows from the associativity of ";" and its monotonicity w.r.t. refinement. We state the monotonicity result for what we will call a *sequence of retrenchments* in its generalised, "evolving" form.

Theorem RetSeqE

$$S1 \lesssim T1 \;\text{ w.r.t. }\; (G_\alpha, P1_\alpha(i_1, j_1, u, v, A_1) \longrightarrow G_\beta, C1_\beta(u, v, o_1, p_1, A_1))$$
$$\wedge\; S2 \lesssim T2 \;\text{ w.r.t. }\; (G_\beta, P2_\beta(i_2, j_2, u, v, A_2) \longrightarrow G_\gamma, C2_\gamma(u, v, o_2, p_2, A_2))$$
$$\wedge\; I \wedge G_\alpha \wedge J$$
$$\wedge\; \exists \alpha, A_1 \bullet P1 \wedge [T1]\neg\, [S1]\neg\, (G_\beta \wedge P2_\beta)$$
$$\wedge\; \{\mathsf{trm}(S1;S2)(u, i_1, i_2) \vee \forall u'_1 \bullet \exists v'_1 \bullet \{I(u'_1)$$
$$\quad \Rightarrow G_\beta(u'_1, v'_1) \wedge J(v'_1) \wedge \exists j_2, A_2 \bullet P2_\beta(i_2, j_2, u'_1, v'_1, A_2) \wedge \mathsf{trm}(T2)(v'_1, j_2)\}\}$$
$$\wedge\; \mathsf{trm}(T1;T2)$$
$$\vdash \mathsf{trm}(S1;S2) \wedge [T1;T2]\neg\, [S1;S2]\neg\, (G_\gamma \vee C2_\gamma)$$

More succinctly, we say

$$S1;S2 \lesssim T1;T2 \quad \text{w.r.t.} \quad (G_\alpha, P_\beta{}^; \longrightarrow G_\gamma, C2_\gamma)$$

where $P_\beta{}^;$ is defined by the fourth through sixth lines of the hypotheses of RetSeqE.

The sequence of retrenchments result has three conjuncts in its WITHIN clause (lines 4-6 above). The first requires that the first-step WITHIN constraint P_1 be witnessable for some α, A_1. The second, satisfying intuition, requires the first-step retrenchment to be able to establish both the RETRIEVES clause and the second-step WITHIN clause P_2: the second step must be guaranteed to start before sequenced retrenchment can be established. The third conjunct is a weakened termination hypothesis, which means that the sequenced termination result (that is, inference of abstract sequenced termination) is nontrivial. Note that the composite WITHIN clause $P_\beta{}^;$ has only β as precision parameter. The sequenced concession is identical to the second-step concession.

Simple applications of sequence of retrenchment in the evolving-precision arithmetic example serve to demonstrate the utility of an evolving version of retrenchment. Thus, without the stronger P'_+ assumption at either step, retrenchment of sequence of additions is defined, losing precision at 1 per operation step, i.e. losing 2 for a 2-step sequence:

$$+_a;+_a \lesssim +_c;+_c \quad \text{w.r.t.} \quad (G_n, P^;_{++} \longrightarrow G_{n+2}, \mathsf{false}) \qquad (1)$$

...where

$$P^;_{++}\ (i_1, i_2, j_1, j_2) \mathrel{\widehat{=}} P(i_1, j_1) \wedge P(i_2, j_2) \wedge [+_c]\neg\ [+_a]\neg\ G_{n+1}$$

This is easy to see by application of RetSeqE: from discussion of the example earlier we have the hypotheses

$$+_a \lesssim +_c \quad \text{w.r.t.} \quad (G_n, P \longrightarrow G_{n+1}, \mathsf{false})$$
$$\wedge\ +_a \lesssim +_c \quad \text{w.r.t.} \quad (G_{n+1} \longrightarrow G_{n+2}, \mathsf{false})$$
$$\wedge\ I \wedge G_n \wedge J \wedge P(i_1, j_1)$$

Result (1) follows from further assuming the extra hypothesis about establishment of intermediate-state $G_{n+1} \wedge P(i_2, j_2)$, which reduces (because intermediate-state P has new input variables only) to

$$P(i_2, j_2) \wedge [+_c]\neg\ [+_a]\neg\ G_{n+1}$$

Of course we already have the second conjunct of this.

Returning to the control system analogy, this represents two steps with sensor (and precision) loss at each step.

Finally, leaving the working as an exercise for the reader, we see that, if addition loses precision 1 under P, and subtraction gains precision 1 under P'_{-n}, then overall precision is maintained by the sequence

$$+_a;-_a \lesssim +_c;-_c \quad \text{w.r.t.} \quad (G_n, P^;_{+-} \longrightarrow G_n, \mathsf{false}) \qquad (2)$$

If the specifier has some freedom in choosing the permutation of operation sequence at run-time, then some precision optimisation is possible. This effect cannot be seen in this simple example, but is well known in the Numerical Analysis of floating point, e.g. [Atk89].

The example does, however, show that the choice of sequence permutation determines the composite WITHIN clause. It is not obvious *a priori* which permutation delivers the weakest (and thus most desirable architecturally) WITHIN clause, or even whether such clauses are comparable by implication.

The question of associativity of the sequencing of retrenchments must be mentioned here: it transpires that the composite retrenchment is dependent on association order of component retrenchments. However, it can be shown that the right-associated WITHIN clause is strictly stronger than the left [Pop00], a fact which is useful architecturally. It is interesting that an appropriate counterexample for associativity is Milner's classic example of late resolution of non-determinism, used to demonstrate the incompleteness of forward simulation as a proof method for refinement in [dRE98]. This is a technical matter which is pursued in [Pop00].

7 Conclusion

The radiation dosimetry example is illuminating. The simple version of the problem presented in section 3.1 showed the need for two retrenchment steps to distinguish between two distinct design approximations: fixed-precision arithmetic and circle-segment-approximation for scatter calculation. The general dosimetry problem affords more layers of modelling and raises other architectural issues.

Classical models for dose absorbtion are usually based on the Boltzmann Transport Equation [BFM67], a three-dimensional nonlinear integro-differential equation describing radiation energy density. There are no known analytic solutions and numerical approximation is not straightforward. In practice, radiation physicists start from a simpler empirical model based on weighted sums of exponentially attenuating components, which is developed heuristically by curve-fitting to experimental data. In principle this represents the first retrenchment step. This step constitutes the belief that the Transport Equation does indeed describe the physical situation demonstrated in the experiment, and therefore that, within suitable error bounds, the empirical model will match the abstract one. Although in radiation dosimetry this level of verification is unlikely to happen, it is in fact typical of the extra-mathematical form of verification required moving from the physical/ continuous-mathematical description of a real-world system to a model closer to software implementation.

The next retrenchment step takes the empirical model, still using continuous mathematics, to a discretised model involving summations and finite algorithms over finite sets of limited precision values. As seen in previous discussion, architectural separation of concerns will usually require more than one retrenchment to reach such a model. At that stage the designer is finally in a sufficiently finite, discrete, strongly defined world to be able to refine to types and algorithms to meet the usual requirements of memory utilisation, speed etc.

To return to applications, modern control systems are of course more complex than the analogy of section 5 suggests: in general, more is modelled than state "now", i.e. at the last cycle or interrupt of sensor readings. PID (proportional, integral, differential) controllers [Rav77] require recent state history information, in order to model integral (summation) and differential (rate-of-change) properties of dynamic state variables. Thus sensor recovery at a given instant does not imply complete instantaneous recovery of the real-world information delivered by that sensor: a period of time is required to build a picture of integral and differential properties of the variable being sensed. Recovery of a sensor implies return over a number of steps (while state history information from that sensor is reassembled), through a gradually improving mode of operation, to an appropriate recovered mode.

This relates to the question of history and prophecy variables, classically examined in the context of refinement in [AL91]. Some syntactic manipulations facilitate the strengthening of the primitive notion of operation sequence in B, in order to make intermediate-state variables explicit in postconditions. Such

a notion of *strong sequence* then provides a vehicle for specification with history variables as required by control systems; this work is reported in [Pop00]. Of course it may be that process-algebraic notations such as suggested by [TS99] may be more succinctly used in conjunction with B to describe such systems.

Much further work is evidently required to express retrenchment in popular formal frameworks, such as Z and VDM, as well as notations for concurrent and distributed systems, in order that researchers may examine and exploit the notion in applications. Such exploration and evaluation should take the form of case studies, both academic and industrial, to demonstrate the utility of retrenchment in extending formal verification to parts of safety-critical system developments that refinement can't reach. Such work will of course be influenced by the state of formalisation of continuous and numerical mathematics, and by the level of integration of such formalisation with verification tools and notations used in the Formal Methods community.

References

[Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991.

[Atk89] K.E. Atkinson. *An Introduction to Numerical Analysis.* Wiley, 1989.

[Bac81] R.J.R. Back. On correct refinement of programs. *J. Comp. Sys. Sci.*, 23:49–68, 1981.

[BB98] R.J.R. Back and M. Butler. Fusion and simultaneous execution in the refinement calculus. *Acta Informatica*, 35:921–949, 1998.

[BFM67] G.J. Brownell, J.J. Fitzgerald, and F.J. Mahoney. *The Mathematical Theory of Radiation Dosimetry.* Gordon and Breach, 1967.

[Bli81] A. Blikle. The clean termination of iterative programs. *Acta Informatica*, 16:199–217, 1981.

[BP98] R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. In D. Bert, editor, *Proc. B'98:Recent Advances in the Development and Use of the B Method*, volume 1393 of *LNCS*, Montpellier, France, April 1998. Springer.

[BP99a] R. Banach and M. Poppleton. Retrenchment and punctured simulation. In K. Araki, A. Galloway, and K. Taguchi, editors, *Proc. IFM'99:Integrated Formal Methods 1999*, pages 457–476, University of York, June 1999. Springer.

[BP99b] R. Banach and M. Poppleton. Sharp retrenchment, modulated refinement and punctured simulation. *Formal Aspects of Computing*, 11:498–540, 1999.

[BP00] R. Banach and M. Poppleton. Retrenchment, refinement and simulation. 2000. submitted for publication.

[BvW89] R. J. R. Back and J. von Wright. Refinement calculus part I: Sequential nondeterministic programs. In de Roever and Rozenberg, editors, *Proc. REX Workshop, Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, pages 42–66. Springer, 1989.

[CH79] D. Coleman and J.W. Hughes. The clean termination of pascal programs. *Acta Informatica*, 11:195–210, 1979.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[dRE98] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* Cambridge, 1998.

[HHS86] J. He, C.A.R. Hoare, and J.W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *ESOP86: European Symposium on Programming*, volume 213 of *LNCS*. Springer, 1986.

[HHS87] C.A.R. Hoare, J. He, and J.W. Sanders. Prespecification in data refinement. *Information Processing Letters*, 25:71–76, 1987.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[HW94] A. R. Hounsell and J. M. Wilkinson. Dose calculations in multi-leaf collimator fields. In *Proceedings of the XIth International Conference on the use of Computers in Radiation Therapy*, Manchester, UK, March 1994.

[JC76] H.E. Johns and J.R. Cunningham. *The Physics of Radiology.* Charles C. Thomas, 1976.

[Kha94] F.M. Khan. *The Physics of Radiation Therapy.* Williams and Wilkins, 1994.

[Mor94] J. M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9:287–306, 1994.

[Nei90] D.S. Neilson. *From Z to C: Illustration of a Rigorous Development Method.* PhD thesis, Oxford University Programming Research Group, 1990. Technical Monograph PRG-101.

[Owe93] O. Owe. Partial logics reconsidered: A conservative approach. *Formal Aspects of Computing*, 3:1–16, 1993.

[PB99] M. Poppleton and R. Banach. Retrenchment: Extending the Reach of Refinement. In *ASE'99: Fourteenth IEEE International Conference on Automated Software Engineering*, Cocoa Beach, Florida, October 1999. IEEE Computer Society Press.

[Pop00] M.R. Poppleton. *Formal Methods for Continuous Systems.* PhD thesis, Department of Computer Science, University of Manchester, 2000. in preparation.

[Rav77] F.H. Raven. *Automatic Control Engineering.* McGraw-Hill, third edition, 1977.

[SCW98] S. Stepney, D. Cooper, and J. Woodcock. More powerful Z data refinement: Pushing the state of the art in industrial refinement. In J.P. Bowen, A. Fett, and M.G. Hinchey, editors, *ZUM'98:The Z Formal Specification Notation*, volume 1493 of *LNCS*, pages 284–307, Berlin, Germany, 1998. Springer.

[Spi93] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, second edition, 1993.

[TS99] H. Treharne and S. Schneider. Using a process algebra to control b operations. In K. Araki, A. Galloway, and K. Taguchi, editors, *IFM'99:Proceedings of the First International Conference on Integrated Formal Methods*, pages 437–456, University of York, June 1999. Springer.

[WD96] J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof.* Prentice-Hall, 1996.