Engineering and Theoretical Underpinnings of Retrenchment

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

Abstract. Refinement is reviewed in a partial correctness framework, highlighting in particular the distinction between its use as a specification constructor at a high level, and its use as an implementation mechanism at a low level. Some of its shortcomings as specification constructor at high levels of abstraction are pointed out, and these are used to motivate the adoption of retrenchment for certain high level development steps. Basic properties of retrenchment are described, including a justification of the operation PO, simple examples, simulation properties, and compositionality for both the basic retrenchment notion and enriched versions. The issue of framing retrenchment in the wide variety of correctness notions for refinement calculi that exist in the literature is tackled, culminating in guidelines on how to 'brew your own retrenchment theory'. Two short case studies are presented. One is a simple digital redesign control theory problem, the other is a radiotherapy dose calculation problem, addressed from a more architectural level.
Keywords. Refinement, Retrenchment, Simulation, Compositionality, Partial and Total Correctness, Digital Redesign, Radiotherapy.

1 Introduction

One of the most startling things about the present state of the interaction between formal methods of software development and the practice of software development, particularly as it applies to the development of software that controls real physical apparatus, is the mutual incomprehension that persists between researchers and practitioners. One of the worst aspects of this, is the perceived inadequacy of refinement techniques in the face of the demands of real applications, a deficiency simultaneously denied by researchers and held as selfevident by practitioners. The main motivation for the subject of this paper, retrenchment, is to help to assuage this dissonance. Retrenchment, which was first introduced in [Banach and Poppleton (1998)] in the specific context of the B-Method [Abrial (1996a), Wordsworth (1996), Lano and Haughton (1996), Sekerinski and Sere (1998)], is a more liberal formal technique, based on the main ideas of refinement, and is intended so that some pairs of models which cannot be related within a development by refinement alone, can nevertheless be included within a formal development by its help.

This paper presents the retrenchment concept from scratch for a wider range of notions of system correctness (than specifically the B-Method). It starts by observing that over the years, refinement has been used not only as a method of implementation which guarantees the properties captured in a specification, but also as a method of introducing structure and detail into a system development. When this detail addresses requirements, it can have the effect of clouding the distinction between specification and implementation because the 'real' specification is not in fact the most abstract model in the development. Of course in many of the formal methodologies that

have been developed for expressing refinement and calculating with it, see for example [Back (1981), Back (1988), Back and von Wright (1989), von Wright (1994), Morris (1987), Morgan (1994), Back and von Wright (1998)], these questions of emphasis are not always at the fore.

Thinking in a less formal vein, many examples developed within notations like Z or VDM [Spivey (1993), Hayes (1993), Jones (1990), Jones and Shaw (1990)], also use the concept of refinement, but in a looser sense, to describe development steps which do not strictly possess the properties of the formal refinement notion as it appears in the earlier citations. That these development steps are useful stages in the realisation of systems is indisputable; certainly so from a systems engineering point of view. They therefore give rise to the need for a richer formal development notion, one that is capable of capturing such steps in a formal way, thus bringing them into the formal fold. Retrenchment is a contribution to this need.

In this paper we start in Section 2 by discussing the various roles played by refinement in the development process. We distinguish particularly refinement as specification constructor from refinement as implementation tool, noting that the distinction is often subtly blurred and that abstract models are often reverse engineered from more concrete ones. We illustrate our ideas with examples and pause to define various simulation theoretic notions. Here as in the majority of the paper, we work in a simple transition system framework. In Section 3 we illustrate some of the problems that can arise during development steps if we adhere strictly to the formal definition of refinement as sole method of passing from more abstract to more concrete models. Having presented our motivations, Section 4 defines retrenchment itself, justifying the key proof obligation, presenting a simple example, revisiting the simulation theoretic notions of Section 2, and proving composability of retrenchments — pointing out that in fact composability extends to a much wider range of retrenchment-like notions than the one focused on in this paper. We also review how retrenchment relates to other work in the literature that addresses the limitations imposed by the 'refinement straitjacket'. In Section 5 we examine policies for the initiation and termination of operations, variations in the detail of which are the source of an enormous variety of distinctions between formal program development frameworks. These can give rise to what we call the 'Cynical Refinement Practitioner's Manifesto', an elaboration of the reverse engineering attitude already mentioned. Rather than attempt an exhaustive coverage of retrenchment for each separate policy, we give guidelines that will enable the adherents of any particular scheme to 'brew their own' definition of retrenchment. In Section 6 we consider case studies. After some general discussion of the relationship between applied mathematics and the formal schemes usually found in formal program development frameworks, we discuss a simple continous-discrete control theory problem in detail, and outline the architecture of a retrenchment based approach to radiotherapy dose calculations. Section 7 concludes, and indicates future directions.

2 Refinement, and Requirements

Back in the days of [Wirth (1971), Dijkstra (1972), Hoare (1972)], life was simple. It was clear what was meant by refinement. It was a process whereby a piece of abstract program (in some context) could be replaced by a piece of more concrete program (in the same context), without the observer being any the wiser. The argument

went, that given appropriate sufficient conditions, it could be *proved* that the observer would be none the wiser, so convenient sets of sufficient conditions got adopted as particular paradigms for refinement.

After some time, almost imperceptably, life got more complicated. The process of refinement in the preceding sense, typically invoves the incorporation/adoption of lower level detail into the lower level model. Indeed, one of the much vaunted strengths of the refinement technique is its ability to delay the consideration of lower level detail in the development process. However, the more general question then arises, as to what extent this lower level detail forms part of the original requirements of the system, and thus, to what extent the original abstract model deserves to be called a specification of it. For if the lower level detail indeed addresses system requirements, then the abstract model cannot have been a complete specification (assuming that the purpose of a specification is to express a system model that captures *all* the requirements, at the highest possible level of abstraction).

The practice of refinement thus became muddied by a lack of clarity as to what the relationship between requirements and the abstract model was supposed to be; the more so as research was heavily concentrated on the technicalities of formal specification, to the detrement of requirements considerations. Indeed in many textbook-scale examples of refinement, the more abstract models are in fact reverse engineered from the more concrete ones (perhaps subconsciously), by a process of forgetting the kind of detail that experience has taught us can be elegantly reintroduced via the mechanisms that refinement puts at our disposal.

Going by the authors' personal experience, it is disturbing how quickly one gets seduced into this mode of thinking. Upon starting work with formal refinement, when everything is still new, the selection of what detail is to be expressed at the abstract level can often seem jarring when considered against what one might 'naturally' take to be the most abstract aspects of the system. But this feeling passes surprisingly quickly. Before one knows it, one has fallen into the comfortable habit of choosing just the 'right' aspects of the system to include at the abstract level; a choice made so that the remaining ones 'magically' yield to the refinement technique. Soon this practice is second nature, and one has forgotten that there ever was any discomfort.

2.1 Black, Grey, and Glass Boxes

Let us examine this issue in a little more detail via examples. We start by outlining the framework in which we will work. We will want to discuss relationships between an abstract system *Abs* and a concrete one *Conc*; in general these will just be two adjacent systems in a development hierarchy. At the abstract level, there will be a set of operation names Ops_A, with typical element Op_A. These operations will work on a state space U, having typical element u. For an $Op_\mathrm{A} \in \mathsf{Ops}_\mathrm{A}$, there will also be input and output spaces $\mathsf{I}_{Op_\mathrm{A}}$ and $\mathsf{O}_{Op_\mathrm{A}}$ with typical elements i, o respectively (we will suppress without causing any confusion, the anticipated subscripts on i and o to indicate which Op_A they belong to). Primes, indeces and other decorations will be used to distinguish different elements of the same space. Thus initial states will satisfy the property $Init_\mathrm{A}(u')$. For the time being, we will work in a transition system framework. So an operation Op_A will be defined by its transition or step relation, written $stp_{Op_\mathrm{A}}(u, i, u', o)$, consisting of steps u -(i, Op_A, o)-› u', where u and u' are the before- and after- states, and i and o are the input and output values. Only steps

that initiate and terminate successfully enter the discourse at this point; we will return to other aspects of computational steps in Section 5. An execution fragment of the abstract system Abs is a finite or infinite sequence of contiguous steps from the collection of abstract step relations $\bigcup\{stp_{Op_A} \mid Op_A \in \mathsf{Ops}_A\}$, which we typically write as $[\ u_0\ \text{-}(i_0, Op_{A,0}, o_1)\text{-}\!\rightarrow\ u_1\ \text{-}(i_1, Op_{A,1}, o_2)\text{-}\!\rightarrow\ u_2 \ldots\ ]$. An execution fragment such that $Init_A(u_0)$ holds is called an execution sequence. An abstract state u is reachable, iff it is the last state of some execution sequence.

At the concrete level we have a similar setup. The operation names are $Op_C \in \mathsf{Ops}_C$. States are $v \in \mathsf{V}$, inputs $j \in \mathsf{J}$, outputs $p \in \mathsf{P}$. Initial states satisfy $Init_C(v')$. Transitions are $v\ \text{-}(j, Op_C, p)\text{-}\!\rightarrow\ v'$, which are elements of the step relation $stp_{Op_C}(v, j, v', p)$.

In keeping with the overwhelming majority of applications, in this paper we stay within a forward simulation formulation of refinement. See [de Roever and Engelhardt (1998)] for a comprehensive survey of refinement from both forward and backward simulation perspectives, and under various notions of correctness. To this end, we assume there is a bijection between abstract and concrete operation names, which defines which concrete operation refines which abstract one. For simplicity we will assume this bijection is an identity, giving rise to some overloading of nomenclature. This means that the 'A' and 'C' subscripts on Op_A and Op_C are meta-level tags, and we will suppress such subscripts when it makes sense to do so.

In this simple framework, refinement is primarily expressed by a retrieve relation between abstract and concrete states $G(u, v)$. This has to satisfy two properties, the initialisation and operation proof obligations (POs) respectively. The initial states must satisfy:

$$Init_C(v') \ \Rightarrow\ (\exists\, u' \bullet Init_A(u') \wedge G(u', v')) \tag{2.1}$$

and for every corresponding operation pair Op_A and Op_C, the abstract and concrete step relations must satisfy:

$$\begin{aligned} G(u, v) \wedge stp_{Op_C}(v, j, v', p) \ \Rightarrow\ (\exists\, u', i\ , o \bullet\ & stp_{Op_A}(u, i, u', o) \wedge G(u', v') \wedge \\ & In_{Op}(i, j) \wedge Out_{Op}(o, p)) \end{aligned} \tag{2.2}$$

where In_{Op} is a relation that describes how inputs for Op_A and Op_C relate to each other, and Out_{Op} is a relation that describes how outputs relate. Until further notice, all In_{Op} and Out_{Op} relations will be identities.

We consider a simple example. The abstract system state is a multiset *mset* of natural numbers, and there are two operations *put* and *get* to respectively add an element to *mset* and extract an (arbitrary) element from *mset*. The transition relations for these are thus:

$$mset\ \text{-}(n, put_A)\text{-}\!\rightarrow\ mset \cup \{n\}\ \ ;\ \ mset \cup \{n\}\ \text{-}(get_A, n)\text{-}\!\rightarrow\ mset \tag{2.3}$$

In the usual manner, we can refine the multiset to a sequence *mseq*, with *put* and *get* becoming obvious list manipulation operations:

$$mseq\ \text{-}(n, put_C)\text{-}\!\rightarrow\ mseq \mathbin{@} [n]\ \ ;\ \ n :: mseq\ \text{-}(get_C, n)\text{-}\!\rightarrow\ mseq \tag{2.4}$$

The retrieve relation for these is:

$$G(mset, mseq)\ =\ (\mathrm{mrng}(mseq) = mset) \tag{2.5}$$

where mrng returns the multiset range of a sequence. It is immediate that:

$$Init_A(\emptyset) \ ; \ Init_C([\,]) \tag{2.6}$$

provide a suitable initialisation, and that the PO (2.2) holds.

This example, though tiny, has a noteworthy feature. The concrete system is *fair* in that elements put_C'ed earlier, are inevitably get_C'ed earlier; indeed in any execution of the concrete system, the squence of elements output via get_C is unfailingly a prefix of the squence of elements input via put_C, a property not shared by the abstract system. Is fairness, or the prefix property, a requirement of this system? We didn't say, and the two possible answers bring different interpretations to our minuscule development.

If fairness is not a requirement, then the refinement is an example of a refinement of the traditional kind, in which (2.3) is indeed the specification and (2.4) is (a step towards) an implementation, a black box activity. If it is, then (2.3) could not have been the complete specification since it does not guarantee fairness, and the refinement is merely a step along the way to it. This is a glass box activity, as we would need to make clear to the customer the differing properties of the two models and what role the relationship between them was playing.

We note in passing that in (2.4), the fairness requirement is not being addressed directly within the model, but emerges as a consequence of the properties of the functional description. Indeed if the prefix property were a specific requirement, then (2.4) would precisely capture it as queues are categorical for total orders with the given two operations. However if mere fairness were required (i.e. that every *put*'ed element was eventually *get*'ed sometime), then (2.4) is an instance of implementation bias, since total orders are sufficient but not necessary for mere fairness. To avoid implementation bias, or to express fairness at the abstract level, one would have to step outside the framework we have set up, and resort to temporal logic of one kind or another [Manna and Pnueli (1992), Lamport (1994), Schneider (1997)].

One could address the fairness requirement to a degree yet still avoid embroiling the system description in temporal logic (disregarding whether or not this would be wise) by refining (2.3) in a more complex way. For the sake of the following example only, we will allow the concrete system in the refinement to contain additional, *hidden*, operations, making it a refinement of the action system or superposition refinement kind [Back and Kurki-Suonio (1983), Francez and Forman (1990), Katz (1993), Back and Sere (1996)]. It will also prove useful in making a point below.

Specifically, we will *put* elements into bins using a coarse grained timestamp, and *get* them from the oldest nonempty bin. To assist in this there will be a hidden operation *tick* that increments a natural valued state variable *tickvar* (we suppress the rest of the state for clarity):

$$tickvar \ \text{-}(tick_C)\text{-}\rightarrow \ tickvar{+}1 \tag{2.7}$$

and the rest of the state itself will be a multiset-of-naturals valued map *bins* on the naturals, with the two visible operations given by (using $\lhd$ for relational override):

$$(bins, tickvar) \ \text{-}(n, put_C)\text{-}\rightarrow \ (bins \lhd \{tickvar \mapsto (bins(tickvar){+}\{n\})\}, tickvar)$$
$$;$$

$$(\{0\ldots k\text{–}1\}\times\varnothing \cup \{k \mapsto abin+\{n\}\} \cup \{k+1 \mapsto bins(k+1)\} \ \ldots\ , tickvar)$$
$$\text{-}(get_C,\ n)\text{-}\rightarrow$$
$$(\{0\ldots k\text{–}1\}\times\varnothing \cup \{k \mapsto abin\} \cup \{k+1 \mapsto bins(k+1)\} \ \ldots\ , tickvar) \tag{2.8}$$

The corresponding retrieve relation will be:

$$G(mset, (bins, tickvar)) \ = \ (\Sigma_{\ k\in \mathsf{NAT}}\ bins(k) = mset) \tag{2.9}$$

and the rest can be imagined. Of course this system also displays some implementation bias with respect to a simple fairness requirement, but arguably a bit less than the queue system above. Since there are aspects of the system that we view as not central to the way the fairness requirement is addressed (eg. the details of $tick_C$, and the way the distribution of invocations of $tick_C$ relates to real world time), we could reasonably regard this refinement as a grey box activity, partly transparent and partly opaque. Of course there will also be a refinement of this system to the queue system (augmented by a hidden *tick* operation whose definition would be the identity relation on the state), which we leave to the reader. Summarising, refinement can be used in a variety of ways to address requirements, and the same calculation may be viewed in different lights depending on aspects which are frequently not enunciated clearly.

2.2 Refinement as Specification Constructor

The culmination of the preceding line of thought is that refinement has quietly become (aside from its original purpose), a *specification constructor*, enabling more appropriate specifications to be built from preliminary models that do not in themselves capture all of the requirements of the desired system. There is of course no harm at all in this provided one is honest about what is going on. Indeed in the Specware system [Srinivas and Jullig (1995), Waldinger et al. (1998)], refinement is elevated to a first class specification constructor along with disjoint sum and colimit, inclusion, and parametric instantiation (to mention just the more obvious ones, see also eg. [Ehrig and Mahr (1985), Ehrig and Mahr (1990), Ehrig and Grosse-Rhode (1994), Fiadeiro and Maibaum (1997)]). The confusion arises when one is *not* being sufficiently open about what is going on.

The crucial point is thus to be clear about which model in a refinement hierarchy is the one that is supposed to capture all the requirements, and that thus can serve as a contract between specifier and implementor. We will call this the contracted model. Models above this one in the hierarchy are merely useful preliminaries, and in moving towards the contracted model, refinement is being used as a constructor to enable the gradual accomodation of individual requirements into the contracted model; this is a glass box, or at a least grey box process — the activity of constructing the contracted model should be a transparent one, its details open to inspection by all interested parties, in order to convince all concerned that the right system is being constructed. Models below the contracted model in the hierarchy represent implementation steps, and in refining the contracted model, refinement is being used as a means of achieving implementability on some target system, an essentially black box activity — the users' perspective is already captured within the contracted model, and how this is turned by the implementor into a running system, need not concern the users. Thus these lower level models may enjoy further properties as a result of their more concrete nature, but these properties do not form part of the requirements.

One issue sharply distinguishes the use of refinement in a glass or grey box manner above the contracted model, from its black box use below the contracted model. Below the contracted model I/O signatures must remain unchanged — In_{Op} and Out_{Op} must be identity relations — otherwise how are users to be fooled into believing they are using the abstract model when they are in fact using the concrete one?[1] Above the contracted model there is no such restriction since we are being open about the refinements being done, and In_{Op} and Out_{Op} may be nontrivial relations.

2.3 Simulation Properties

We will now examine some simulation theoretic aspects of refinement. Suppose we have a refinement given by operation names $\mathsf{Ops}_A = \mathsf{Ops}_C$, retrieve relation G, and In_{Op}, Out_{Op} relations for each $Op \in \mathsf{Ops}$. Suppose that $\mathcal{S} = [\ u_0\ \text{-}(i_0, Op_{A,0}, o_1)\text{-}\!\rightarrow u_1\ \text{-}(i_1, Op_{A,1}, o_2)\text{-}\!\rightarrow u_2 \ldots\ ]$ and $\mathcal{T} = [\ v_0\ \text{-}(j_0, Op_{C,0}, p_1)\text{-}\!\rightarrow v_1\ \text{-}(j_1, Op_{C,1}, p_2)\text{-}\!\rightarrow v_2 \ldots\ ]$ are abstract and concrete execution sequences of equal length, either finite or countably infinite, and with the operation names matching for each operation index r. (N.B. In the finite case the states and outputs will have an extra index.) Then we say that $\mathcal{S}$ is a stepwise simulation of $\mathcal{T}$ iff:

$$G(u_r, v_r) \wedge In_{Op_r}(i_r, j_r) \wedge G(u_{r+1}, v_{r+1}) \wedge Out_{Op_r}(o_{r+1}, p_{r+1}) \tag{2.10}$$

holds for each operation index r.

Stepwise simulation will prove to play a pivotal role later in the paper, as the fulcrum of the relationship between refinement and retrenchment.

Proposition 2.3.1 Let *Conc* refine *Abs*. Then every concrete execution sequence $\mathcal{T}$ has a stepwise simulation $\mathcal{S}$.

Proof. This is a trivial induction, with (2.1) providing a base case, and (2.2) the inductive step, constructing $\mathcal{S} = [\ u_0\ \text{-}(i_0, Op_{A,0}, o_1)\text{-}\!\rightarrow u_1\ \text{-}(i_1, Op_{A,1}, o_2)\text{-}\!\rightarrow u_2 \ldots\ ]$ from $\mathcal{T}$ as required, with $\mathcal{S}$ and $\mathcal{T}$ of equal length and operation names matching. ☺

Again in the context of a refinement from *Abs* to *Conc*, let us define separately relations on states and transition labels via:

$$\begin{aligned} &\Theta_S(u, v) \;=\; G(u, v) \\ &\Theta_L((i, Op_A, o), (j, Op_C, p)) \;=\; In_{Op}(i, j) \wedge Out_{Op}(o, p) \end{aligned} \tag{2.11}$$

Then we say that the *Abs* transition system strongly simulates the *Conc* transition system iff:

$$Init_C(v') \;\Rightarrow\; (\exists\, u' \bullet Init_A(u') \wedge \Theta_S(u', v')) \tag{2.12}$$

and for all reachable concrete states v:

$$\begin{aligned} &\Theta_S(u, v) \wedge v\ \text{-}(j, Op_C, p)\text{-}\!\rightarrow v' \;\Rightarrow\; (\exists\, u', i\ , o \bullet u\ \text{-}(i, Op_A, o)\text{-}\!\rightarrow u' \wedge \\ &\quad \Theta_L((i, Op_A, o), (j, Op_C, p)) \wedge \Theta_S(u', v')) \end{aligned} \tag{2.13}$$

Of course this is nothing but a trivial restatement of (2.1) and (2.2), but expressed in an automata-theoretic style. Obviously:

1. It may of course be the case that below the contracted model different representations of inputs and outputs may be of use, but the requisite transformations must be performed privately, not at the public interface of the operation.

Proposition 2.3.2 Let *Conc* refine *Abs*. Then there is a strong simulation of *Conc* by *Abs*.

Proceeding further, an interesting perspective on refinement is adapted from logic [van Dalen (1997), Andrews (1986), Hodges (1993)]. A theory T_2 in a formal language $L_2 \supset L_1$ is a conservative extension of a theory T_1 in language L_1, whenever every L_1-theorem in T_2 is already a theorem in T_1 (the reverse implication normally being immediate). We call an extension semiconservative if the reverse implication does not hold. We make an analogy between derivations in logical systems and sequences of execution steps in a transition system, according to the correspondence:

$$\begin{array}{rcll} \text{formula} & — & \text{state} & \\ \text{axiom} & — & \text{initial state} & \\ \text{rule of inference of } T_1\ (T_2) & — & \text{operation of } \mathit{Abs}\ (\mathit{Conc}) & \\ \text{inference step} & — & \text{execution step} & \\ \text{(provable) theory} & — & \text{(accessible) set of states} & (2.14) \end{array}$$

In the context of refinement, the analogy of a semiconservative extension is the ability to provide for each execution sequence of the concrete system starting from v_0 and finishing at v_f, an execution sequence of the abstract system starting from u_0 and finishing at u_f, such that:

$$G(u_0, v_0) \wedge G(u_f, v_f) \tag{2.15}$$

holds. This is a finite simulation property. The analogy is with the semiconservative rather than the conservative property because the rules of inference, normally identical in the two logical theories being considered, have as analogues the operations, which are normally not identical in the abstract and concrete transition systems.

Proposition 2.3.3 Let *Conc* refine *Abs*. Then *Conc* is a semiconservative extension of *Abs*.

The proof is a simple corollary of Proposition 2.3.1.

What we have called the semiconservative extension property is often introduced as an abstract definition of refinement. In this general context, unequal length abstract and concrete execution sequences, where the operation names need not match up (and the concrete system may possess operations, potentially hidden, not present in the abstract one), also play a part. See eg. [Abadi and Lamport (1991), Lamport (1989)]. The notion also forms the approach to refinement espoused in the ASM formalism. See eg. [Börger (1999), Börger (1995a), Börger and Schulte (1998), Börger and Schulte (2000), Börger (1995b)] — in specific examples ad hoc techniques are often required. We will keep to the nomenclature introduced here i.e. the (distinct) concepts of: refinement, stepwise simulation, strong simulation, semiconservative extension, in order to avoid confusion.

3 Shortcomings of Refinement as Constructor

Having accepted that refinement is often used in a glass box manner to help build up structure in the contracted model, in this section we will focus on how refinement can sell system engineers short in their desire to start building specifications at as high a level of abstraction as possible.

3.1 Some Technical Pitfalls

We return to the example of (2.3)-(2.4). Suppose for pragmatic reasons that the maximum size of the concrete sequence was limited to 10. Then we simply take the restriction of (2.4) to sequences with maximum length 10, thus:

$$mseq \text{ -}(n, put_C)\text{-}\rightarrow mseq@[n] \;\; ; \;\; n::mseq \text{ -}(get_C, n)\text{-}\rightarrow mseq \;, \quad \text{where length}(mseq) \leq 9 \tag{3.1}$$

The system (3.1) with the retrieve relation (2.5) and obvious initialisations, is a refinement of (2.3), because its *stp* relation is a subrelation of that of (2.4) and (2.2) is an implication from concrete to abstract steps only. This works because the transition system on concrete sequences of unrestricted length is genuinely a conservative extension of the transition system on sequences of maximum length 10 (through an identity retrieve relation).

However, suppose we wish to develop the system further, to make the operation put_C total on states. We can do this by adjoining to (3.1) the transitions:

$$mseq \text{ -}(n, put_C)\text{-}\rightarrow mseq \;, \quad \text{where length}(mseq) = 10 \tag{3.2}$$

In this case the retrieve relation (2.5) causes the refinement to break down, since if $G(mset, mseq)$ holds with length($mseq$) = 10, then $|\, mset \,| = 10$, but after corresponding steps of (2.3) and (3.2), we have length($mseq$) = 10 while $|\, mset \,| = 11$.

We could attempt to recover a refinement in this case by altering the retrieve relation to something like:

$$G(mset, mseq) \;=\; \begin{cases} \text{mrng}(mseq) = mset \;, & \text{if length}(mseq) \leq 9 \\ \text{mrng}(mseq) \subseteq mset \;, & \text{if length}(mseq) = 10 \end{cases} \tag{3.3}$$

but then the concrete and abstract *get* operations would break the refinement in the length($mseq$) = 10 case (unless we contemplated changing the abstract *get* operation).

Going further, we can aspire to make the operation get_C total on states too, to give users feedback in situations where transitions of (3.1) don't exist:

$$[\,] \text{ -}(get_C, \text{EMPTY})\text{-}\rightarrow [\,] \tag{3.4}$$

Here refinement breaks down again since there is no abstract transition that corresponds to (3.4) at all. Moreover the alteration of the I/O signature implied by (3.4) could only ever be contemplated above the contracted model, and is in any case a little unusual for an In_{Op} relation (since there is no abstract output corresponding to EMPTY), spelling more trouble for refinement.

Since refinement is faring badly with respect to minimal attempts to amplify the design to take into account of reasonable boundary considerations, we may as well do a proper job on these aspects. So we introduce two new concrete states *Uflow* and *Oflow* to represent the underflow and overflow situations. Now the transitions of put_C and get_C are given by (3.1) together with:

$$\begin{aligned} &mseq \text{ -}(n, put_C, \text{FULL})\text{-}\rightarrow Oflow \;, \quad \text{where length}(mseq) = 10 \;\; ; \\ &mseq \text{ -}(get_C, \text{EMPTY})\text{-}\rightarrow Uflow \;, \quad \text{where length}(mseq) = 0 \end{aligned} \tag{3.5}$$

To recover the system once it lands in one of these states we introduce a $reset_C$ operation with transitions:

$$Oflow \text{ -}(reset_C, \text{OK})\text{-}\rightarrow [\,] \;,\; Uflow \text{ -}(reset_C, \text{OK})\text{-}\rightarrow [\,] \tag{3.6}$$

Since we intend the $reset_C$ operation to be only used in response to a situation previously signalled by a FULL or EMPTY output, we do not make it total on the states of the system, (and given the feedback to the user via the FULL and EMPTY, we do not demand totality for put_C and get_C in the *Uflow* or *Oflow* states either). Clearly the system consisting of (3.1), (3.5), (3.6) will not be refinement of the abstract (2.3). Nevertheless the development step from (2.3) to (3.1), (3.5), (3.6) addresses a collection of issues which it is quite reasonable to expect to see in the 'above contracted model' phases of development.

3.2 Aspects of Scale

Obviously the example just discussed is trivially small, and various adjustments could be made in the already mentioned spirit of reverse engineering to get a refinement if need be. But one of the more insidious aspects of the way that refinement interacts with the system engineering process concern system size. Potentially, many things that are, for small systems, at best not at all noticeable or at worst only minor irritations, can become for large systems, real impediments to progress. We highlight two areas.

The first area concerns the glass box side, above the contracted model. Here the usefulness of a formal process for specification construction is closely related to how many of the models that need to be considered in the construction of the system in question, are capable of being encompassed within the formal process: the fewer of them that can be so encompassed, the less the development is assisted by formal underpinnings.

Regarding this point we can see a marked difference between developments that are purely within the discrete domain, and those which must take into account the physical world, with its laws expressed usually in terms of continuous mathematics.

In the discrete world, the mathematics permits direct manipulation of the entities in play. At worst, one can often enumerate such things as sets and the relations between them, and this makes the reassembly of complex systems from more primitive components (if not necessarily intuitively simpler ones) more feasible. Using refinement to build up a complex system from such components is thus a more straightforward prospect in such situations, and the main problem that refinement must overcome in these cases is the unnaturalness of system decomposition which it sometimes forces on developments, a phenomenon often alleviated to some degree according to the discussion immediately preceding Section 2.1.

For developments that involve physical world models, but which ultimately have to culminate in discrete computational systems, the prospects for using refinement throughout the whole development process are much reduced. Typically there is an abstraction gap that is not surmounted by conventional notions of refinement, between thinking and model building at the continuous level, and the corresponding activity at the discrete level. The potential for formal techniques to assist in keeping the whole development process mechanically checked (a criterion that gives a formal de-

velopment process the maximum amount of verifiability, though at quite a price) is thus much reduced, and this can limit the perceived applicability of formal techniques in many engineering application areas.

The second area we highlight concerns the black box side, below the contracted model. Here, the complexity of operations for large systems can be such that the gap between the contracted model and the implementation itself, is fairly small — the contracted model becomes almost a restatement of the implementation in another language. This is particularly noticeable when the implementation code contains a lot of case analysis. Usually the bulk of such case analysis is attributable to system requirements, and thus, even if the case analysis is capable of being introduced stepwise by refinement, it still all belongs in the contracted model. In such cases, the costs of separately producing both the contracted model and the implementation may become unacceptably high. Also the contracted model becomes more difficult to read and to validate. (Of course there is no intrinsic harm in having contracted model and implementation of comparable complexity — it is *always* useful to have more than one perspective on a situation, as everyone who enjoys stereoscopic vision would agree.)

Both areas underline the usefulness of a wider gamut of formal techniques, capable of addressing concerns that are vital higher up the development hierarchy; the first to enable potentially more natural decompositions of models, the second to facilitate a wider gap at the bottom of a development, even if this obscures the precise location of the contracted model for economic reasons. Evidently for small systems these phenomena do not arise.

4 Retrenchment

Having set out our motivations at some length above, we now come to retrenchment itself. The familiar slogan about refinement says that it 'weakens the precondition and strengthens the postcondition'. Since we have argued that refinement sells us short in a number of ways, retrenchment is designed to assert the opposite, i.e. to allow 'strengthening of the precondition and weakening of the postcondition'. It does this by amplifying the relationship between abstract and concrete models, from one expressed by the retrieve relation $G(u, v)$ alone, to one expressed by three relations: firstly the retrieve relation $G(u, v)$, and beyond that the within relation $P_{Op}(i, j, u, v)$ and the concedes relation $C_{Op}(u', v', o, p; i, j, u, v)$. The former is concerned with the precondition strengthening while the latter expresses the postcondition weakening. It is this latter weakening aspect that distinguishes retrenchment most strongly from all previous embellishments of the refinement notion. More generally, the within and concedes relations, and their intended purpose, characterise the retrenchment concept whatever formal framework we might choose to embed it in.

Note that the within relation involves the inputs as well as the states. This allows change of input representation and mixing of state and input information on the pre- side of a transition. Likewise the concedes relation involves the outputs and allows change of output representation and mixing of state and output information on the post- side of a transition; moreover pre- states and inputs may appear too, allowing a richer set of possibilities to be expressed by the concedes relation.

4.1 Basic Concepts of Retrenchment

Now we elaborate retrenchment in our transition system framework. As before we assume abstract and concrete operation name sets Ops_A and Ops_C. This time instead of equality we assume merely an inclusion $\mathsf{Ops}_\mathrm{A} \subseteq \mathsf{Ops}_\mathrm{C}$ so the concrete level may contain additional operations. We are at pains to point out that any such additional operations are *not* regarded as hidden, as they were at the end of Section 2.1.

By analogy with refinement, retrenchment is characterised by two POs. The initialisation PO is just as for refinement:

$$Init_\mathrm{C}(v') \;\Rightarrow\; (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge G(u', v')) \tag{4.1}$$

while the operation PO reads:

$$\begin{aligned} &G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_\mathrm{C}}(v, j, v', p) \;\Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_\mathrm{A}}(u, i, u', o) \wedge (G(u', v') \vee C_{Op}(u', v', o, p; i, j, u, v))) \end{aligned} \tag{4.2}$$

The latter expresses in particular how the within relation strengthens the precondition and the concedes relation weakens the postcondition. Note that it only makes sense for those operation names *Op*, common to both systems. These POs define retrenchment in our framework; this is in contrast with the situation for refinement where the corresponding statements are derived from a more abstract notion. We justify our definition in the following manner.

Let us reexamine refinement for a moment. In our transition system framework, the refinement PO (2.2) has one purpose, i.e. to ensure that no concrete step does anything that is not compatible with some abstract step. The rationale for this is (from a black box perspective) that no user of the system should be able to notice that it is not the abstract system doing the work. The $\forall$*Conc-Op*$\exists$*Abs-Op*… structure of (2.2) makes sure that all concrete steps toe the line.

Retrenchment is different since the abstract and concrete systems are definitely incompatible. The glass box nature of retrenchment implies that the relationship between abstract and concrete systems should be viewed first and foremost as means of comprehensibly constructing the latter. A retrenchment proof obligation ought to reflect this, and the following criteria have influenced the form of (4.2).

Firstly, for generality the PO ought to allow a many-many relationship between those abstract and concrete steps that we might want to regard as related; the $\forall$–$\exists$–… form of (4.2) indeed permits this. Secondly, for ease of mechanical implementation, and also for ease of comparison with and integration with refinement, from among all the possible statements we might believe regarding the relationship between the two systems, we ought to restrict to a standard shape of statement; again the $\forall$–$\exists$–… form of (4.2) supports this. Thirdly, since we want to 'strengthen the precondition', we strengthen the antecedent of the implication in a $\forall$–$\exists$–… -shaped PO by conjoining the within relation *P* to the standard retrieve relation *G*, permitting not only the abstract and concrete before-states to occur, but also abstract and concrete inputs. This allows fine tuning of the before-configurations we wish to speak about above and beyond those identified by the retrieve relation *G* alone. Fourthly, since we want to 'weaken the postcondition', we weaken the consequent in the $\forall$–$\exists$–… -shaped PO by disjoining the concedes relation *C* to the retrieve relation *G*. In *C*, we permit not only the abstract and concrete after-states and outputs to occur, but also the before-

entities for greater expressivity. Most importantly, the disjunction allows deviations from strictly 'refinement-like' behaviour to be expressed.

Of course (4.2) is not the only assertion that satisfies all these criteria. Even if we fix on an overall $\forall$–$\exists$–… form for our PO, there are still two possibilities to consider, namely $\forall$*Abs-Op*$\exists$*Conc-Op*$(G \vee C)$ and $\forall$*Conc-Op*$\exists$*Abs-Op*$(G \vee C)$. To illustrate the advantages of the choice we made, we discuss both of these in turn.

If we examine the $\forall$*Abs-Op*$\exists$*Conc-Op*$(G \vee C)$ form, we must consider four things. Firstly, aside from P and C, this form resembles a refinement proof obligation from concrete to abstract systems too closely. It seems that we would be trying to implement the concrete system using the abstract one, taking us in a direction we do not intend to go. Secondly, the $\forall$*Abs-Op* part forces us to say something about all possible abstract steps. In practice there may be many of these that are irrelevant to the more definitive concrete system, since the abstract system is intended to be merely a simplifying guide to the concrete one (see section 6.2 for an example); the necessity of mentioning them, or specifically excluding them via the P clause, would bring an unwelcome complication. Thirdly, this form does *not* make us say something about all possible concrete steps, limiting its usefulness as a tool for the construction and description of the concrete system. And fourthly, in retrenchment, 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 $\forall$*Abs-Op*$\exists$*Conc-Op*$(G \vee C)$ form.

So we turn to the $\forall$*Conc-Op*$\exists$*Abs-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 at least not detrimental. Secondly, we *must* say something about all concrete steps, which helps to enhance the PO as a mechanism for the construction and description of the concrete system, and which we therefore regard as beneficial. In particular we must consider for any concrete step, whether it is: (a) excluded from consideration because P is not validated (or that we ought to ensure that P is constructed in such a way that this is the case); (b), included but requires essential use of C to satisfy the PO; (c), included but does not require C. The third point follows on from (c). In realistic practical cases, there may well be substantial portions of the state and I/O spaces in which it is sufficient for P and C to be trivial. In such places the truth of the refinement PO, follows from the truth of the retrenchment PO. When this arises, we are justified in viewing retrenchment as being 'like refinement except round the edges', encouraging us to employ retrenchment in situations where a refinement development step works 'most of the time' i.e. for the majority of the state and I/O spaces, but not quite everywhere. The identity of the refinement and retrenchment initialisation POs further enhances this view.

4.2 A Simple Example

We return to the example we left at the end of Section 3 wherein refinement foundered and show how it fits conveniently enough into the retrenchment framwork. We recall:

$$\{put_{\mathrm{A}}, get_{\mathrm{A}}\} = \mathsf{Ops}_{\mathrm{A}} \subseteq \mathsf{Ops}_{\mathrm{C}} = \{put_{\mathrm{C}}, get_{\mathrm{C}}, reset_{\mathrm{C}}\}$$

$$\mathsf{U} = \mathcal{P}(\mathsf{NAT}),\ \mathsf{I}_{put_{\mathrm{A}}} = \mathsf{NAT},\ \mathsf{O}_{put_{\mathrm{A}}} = \varnothing,\ \mathsf{I}_{get_{\mathrm{A}}} = \varnothing,\ \mathsf{O}_{get_{\mathrm{A}}} = \mathsf{NAT}$$

$$\mathsf{V} = \{ll \in \mathrm{seq}(\mathsf{NAT}) \mid \mathrm{length}(ll) \leq 10\} \cup \{Uflow, Oflow\},$$

$$\mathsf{I}_{put_C} = \mathsf{NAT},\ \mathsf{O}_{put_C} = \{\text{FULL}\},\ \mathsf{I}_{get_C} = \varnothing,\ \mathsf{O}_{get_C} = \mathsf{NAT} \cup \{\text{EMPTY}\}$$
$$\mathsf{I}_{reset_C} = \varnothing,\ \mathsf{O}_{reset_C} = \{\text{OK}\} \tag{4.3}$$

Reprising the transitions:

$$mset\ \text{-}(n, put_A)\text{-}\rightarrow mset \cup \{n\}\ \ ;\ \ mset \cup \{n\}\ \text{-}(get_A, n)\text{-}\rightarrow mset \tag{4.4}$$

and

$$\begin{array}{l} mseq\ \text{-}(n, put_C)\text{-}\rightarrow mseq@[n]\ ,\ \text{where length}(mseq) \leq 9 \\ mseq\ \text{-}(n, put_C, \text{FULL})\text{-}\rightarrow Oflow\ ,\ \text{where length}(mseq) = 10\ \ ; \\ n::mseq\ \text{-}(get_C, n)\text{-}\rightarrow mseq\ ,\ \text{where length}(mseq) \leq 9 \\ mseq\ \text{-}(get_C, \text{EMPTY})\text{-}\rightarrow Uflow\ ,\ \text{where length}(mseq) = 0\ \ ; \\ Oflow\ \text{-}(reset_C, \text{OK})\text{-}\rightarrow [\,]\ ,\ \ Uflow\ \text{-}(reset_C, \text{OK})\text{-}\rightarrow [\,] \end{array} \tag{4.5}$$

The retrieve relation will be the original and transparent:

$$G(mset, mseq)\ =\ (\text{mrng}(mseq) = mset) \tag{4.6}$$

with the two values *Uflow*, *Oflow* being outside the range of G. The initialisations are once more:

$$Init_A(\varnothing)\ \ ;\ \ Init_C([\,]) \tag{4.7}$$

It remains to give the within and concedes relations for put_C, get_C. These are:

$$\begin{array}{l} P_{put}(i, j, mset, mseq)\ =\ (i = j \wedge mseq \notin \{Uflow, Oflow\})\ \ ; \\ C_{put}(mset', mseq', o, p; i, j, mset, mseq)\ = \\ \quad (p = \text{FULL} \wedge mseq' = Oflow \wedge \text{length}(mseq) = 10 \wedge mset' = mset \cup \{i\}) \\ P_{get}(i, j, mset, mseq)\ =\ (mseq \notin \{Uflow, Oflow\} \wedge \text{length}(mseq) \neq 0)\ \ ; \\ C_{get}(mset', mseq', o, p; i, j, mset, mseq)\ =\ \mathsf{false} \end{array} \tag{4.8}$$

We remark first of all that these are by no means unique. For instance we could omit the terms $mseq \notin \{Uflow, Oflow\}$ from P_{put} and P_{get} as they are are a consequence of the truth of G, which is assumed to hold anytime either of P_{put} or P_{get} is employed in the retrenchment PO. Likewise we could omit various of the clauses from C_{put} without destroying the truth of the PO. Retaining all these clauses however is more informative from a design description point of view. Note also that if we strengthened P_{put} with the clause $\text{length}(mseq) \leq 9$ then we could have reduced C_{put} to false; we would have excluded from consideration the part of put_C that behaves badly. Since the concession trivialises in this case and the consequent of the retrenchment PO is no longer nontrivially disjunctive, we are justified in regarding the retrenchment as having reduced to a form of refinement. This illustrates our contention above that retrenchment is usefully imagined as being 'like refinement except round the edges'. Note the different behaviour of the retrenchment PO at the points that get_C and put_C are about to behave badly. When $\text{length}(mseq) = 10$, both concrete and abstract *put* operations have something to do; the concedes relation describes the incompatibilities that ensue. However when $\text{length}(mseq) = 0$, the abstract *get* operation has no step; all that the retrenchment PO can do is to exclude those situations from consideration. It is not the job of the retrenchment PO to speak about parts of either system that have no counterpart in the other one.

4.3 Simulations and Retrenchment

The discussion and example above aimed to justify heuristically a PO that is simple and convenient to mechanise and to use in real designs. We now turn to simulation theoretic aspects of retrenchment and revisit the ideas of Section 2.3. We first note the differences between the refinement PO (2.2) and the retrenchment PO (4.2). Obviously the fact that the consequent of (4.2) is disjunctive means that unlike refinement, retrenchment does not guarantee to reestablish the retrieve relation. Moreover in (2.2), the abstract input i and its relationship $In_{Op}(i, j)$ with the concrete input j is infered in the consequent, implying that In_{Op} must be onto all of J, or at least onto that part of J which could ever arise in practice[2] (otherwise we could not assert the consequent as stated). By contrast in (4.2), the input i and its relationship $P_{Op}(i, j, u, v)$ with the concrete world occurs free in the antecedent, allowing us to circumscribe the relationship between abstract and concrete worlds as much as we like. These features are essential in affording the flexibility needed for describing certain development steps, but act against the straightforward derivation of simulation results of the kind that hold for refinement. The latter require more effort.

Once more the key notion is stepwise simulation. In the context of a retrenchment given by families of operations $\mathsf{Ops}_\mathrm{A} \subseteq \mathsf{Ops}_\mathrm{C}$, a retrieve relation G, and within and concedes relations P_{Op} and C_{Op} for $Op \in \mathsf{Ops}_\mathrm{A}$, an abstract execution sequence $\mathcal{S} = [\, u_0 \text{-}(i_0, Op_{\mathrm{A},0}, o_1)\text{-}\!\rightarrow u_1 \text{-}(i_1, Op_{\mathrm{A},1}, o_2)\text{-}\!\rightarrow u_2 \ldots]$ is a stepwise simulation of a concrete execution sequence $\mathcal{T} = [\, v_0 \text{-}(j_0, Op_{\mathrm{C},0}, p_1)\text{-}\!\rightarrow v_1 \text{-}(j_1, Op_{\mathrm{C},1}, p_2)\text{-}\!\rightarrow v_2 \ldots]$ iff: they are of equal length (either finite or countably infinite), the operation names at each operation index match, and for each operation index r we have:

$$\begin{aligned} &G(u_r, v_r) \wedge P_{Op_r}(i_r, j_r, u_r, v_r) \wedge \\ &\quad (G(u_{r+1}, v_{r+1}) \vee C_{Op_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}; i_r, j_r, u_r, v_r)) \end{aligned} \tag{4.9}$$

Dropping the $Init_\mathrm{A}(u_0)$, $Init_\mathrm{C}(v_0)$ requirements gives us stepwise simulation for execution fragments. Note that (4.9) requires that for each operation index r, $G \wedge P_{Op_r}$ holds regardless of whether C_{Op_r} holds from a preceding step or not. Thus since the result of a step guarantees only $G \vee C_{Op_r}$ we cannot prove the existence of a stepwise simulation on the basis of a retrenchment alone: we need additional assumptions (as we would also do in refinement if we moved $In_{Op}(i, j)$ to the antecedent in (2.2)).

Let us say that a concrete state v is forward non simulable (FNS) iff for all j, u, i, v', the antecedent of (4.2) is false. We will call an execution fragment (or sequence) FNS iff its final state is FNS. Let Conc^n be the set of FNS concrete execution fragments containing n steps each element of which has at least one stepwise simulation. Let ${}^n\mathit{Conc}$ be the set of concrete execution sequences containing n steps each element of which has at least one stepwise simulation. Let $\mathit{Conc}_n = \mathit{Conc}^n \cap {}^n\mathit{Conc}$. The relationships between these are displayed in (4.10).

In (4.10) the notation $\mathit{Conc}^n \gg_n \mathit{Conc}^{n+1}$ means that the n-step suffixes of sequences in Conc^{n+1} are in Conc^n and ${}^n\mathit{Conc} \gg_n {}^{n+1}\mathit{Conc}$ means that the n-step prefixes of sequences in ${}^{n+1}\mathit{Conc}$ are in ${}^n\mathit{Conc}$. When $n \to \infty$, ${}^\omega\mathit{Conc}$ is the set of stepwise simulable infinite execution sequences, while Conc^ω is the set of stepwise simulable

2. We will ignore this subtlety in future.

$$\begin{array}{ccccccc}
Conc^{\,0} & \gg_0 & Conc^{\,1} & \gg_1 & Conc^{\,2} & \gg_2 & \cdots \\
\cup\!| & & \cup\!| & & \cup\!| & & \\
Conc_{\,0} & & Conc_{\,1} & & Conc_{\,2} & \cdots & \\
|\!\cap & & |\!\cap & & |\!\cap & & \\
{}^{0}Conc & {}_{0}\gg & {}^{1}Conc & {}_{1}\gg & {}^{2}Conc & {}_{2}\gg & \cdots
\end{array} \tag{4.10}$$

infinite execution fragments stretching back into the infinite past and stopping at a specific point in the present. Thus $Conc^{\,\omega} \cap {}^{\omega}Conc$ is empty.

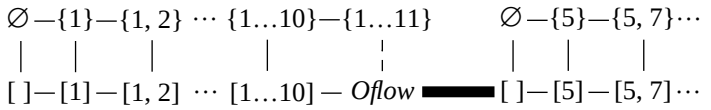
The study of stepwise simulation in retrenchment is the study of the nonemptiness of the various *Conc* sets. As indicated previously, nothing can be said without additional assumptions. Even in the trivial case where all the C_{Op} are empty relations, we need some assumption about the P_{Op} such as:

$$G(u, v) \Rightarrow \bigwedge_{Op \in \mathsf{Ops}_A} (\forall\, j_{Op} \exists\, i_{Op} \bullet P_{Op}(i_{Op}, j_{Op}, u, v))) \tag{4.11}$$

which ensures that given G, no concrete input precludes the existence of a suitable abstract input that fulfills the antecedents of (4.2). (Making the P_{Op} universal relations is hopelessly unrealistic here — it would say that *any* abstract input matched *any* concrete one.) With a proviso like (4.11), a simple inductive proof of the existence of a stepwise simulation is possible. Evidently more restricted asumptions lead to more narrowly focused results.

In general the assumptions needed are either generic and pertain to a class of systems, or specific and come from the detailed properties of an individual system. In the former case, when the generic properties are such that they hold equally for the before and after states of a transition, paraphrasing stepwise simulation into strong simulation becomes realistic, as happened in Section 2.3. And from stepwise simulation, deriving a semiconservative extension is easy. Some examples of these approaches can be found in [Banach and Poppleton (2000a)], worked out for the B-Method.

The fact that an FNS state need not be terminal in the concrete transition system means that a concrete execution sequence may continue beyond the point at which it ceases to be simulable. This gives rise to the notion of punctured simulation, in which some but not all of the steps of a concrete execution sequence have abstract counterparts, (obvious candidates being steps performed by operations with names in $\mathsf{Ops}_C - \mathsf{Ops}_A$). A typical scenario for this is when the concrete system contains error recovery operations that are not present in the abstract system. Such a pair of systems may proceed in stepwise simulation for a while, until the concrete system hits an error state (the dissonance between abstract and concrete systems being captured by the concedes relation at that point), whereupon the concrete system embarks upon recovery, a potentially complex process that makes no sense in the abstract system. Upon completion of recovery, the concrete system may once again become simulable, etc. Early results on punctured simulation appear in [Banach and Poppleton (1999a)], once more in the context of the B-Method.



(4.12)

We can illustrate punctured simulation quite nicely using the example of Section 4.2. Consider the situation illustrated in (4.12) where after initialisation, naturals from 1 to 10 are added to *mset* and *mseq*. Upon attempting to add 11, whereas there is no problem at the abstract level, the concrete level goes into the *Oflow* state. Next, the concrete system invokes its $reset_C$ operation in order to reinitialise (the thick transition in (4.12)); this has no counterpart in the abstract system and the simulation breaks down at this point. Once the concrete system has reinitialised it is once again simulable and a simulation can be constructed as shown. Note that there is no connection between the abstract system's $\{1\ldots11\}$ state and the $\varnothing$ state which initiates the next portion of the punctured simulation; it is as if the abstract system had quantum tunnelled to its initial state. This is because we decided to leave out of the abstract system any notion of resetting, to keep its description small. One thing that the abstract system is *not* doing in the gap between $\{1\ldots11\}$ and $\varnothing$, is stuttering [Abadi and Lamport (1991)].

4.4 Composition

In this section we examine the composition properties of retrenchment and show that two succesive retrenchments compose vertically to make another retrenchment. This is but a first step in the integration of retrenchment with other specification constructors, particularly refinement, in order to build a complete algebraic theory, a task beyond the scope of this paper; see eg. [Banach (2000), Banach et al. (2001a)]. We will deal with a sequence of systems Sys_0, Sys_1, … and will assume that there is a retrenchment given by $G_d(u_{d-1}, u_d)$, $P_{Op,d}(i_{d-1}, i_d, u_{d-1}, u_d)$, $C_{Op,d}(u'_{d-1}, u'_d, o_{d-1}, o_d; i_{d-1}, i_d, u_{d-1}, u_d)$, from the more abstract system Sys_{d-1} to the more concrete system Sys_d.

Proposition 4.4.1 Let Sys_0 be retrenched to Sys_1 using G_1, $\{P_{Op,1}, C_{Op,1} \mid Op \in \mathsf{Ops}_0\}$, and Sys_1 be retrenched to Sys_2 using G_2, $\{P_{Op,2}, C_{Op,2} \mid Op \in \mathsf{Ops}_1\}$. Then Sys_0 is retrenched to Sys_2 using retrieve, within, and concedes relations $G_{(1,2)}$, $\{P_{Op,(1,2)}, C_{Op,(1,2)} \mid Op \in \mathsf{Ops}_0\}$, where:

$$G_{(1,2)}(u_0, u_2) \;=\; (\exists\, u_1 \bullet G_1(u_0, u_1) \wedge G_2(u_1, u_2)) \tag{4.13}$$

$$\begin{aligned} &P_{Op,(1,2)}(i_0, i_2, u_0, u_2) \;= \\ &\qquad (\exists\, u_1, i_1 \bullet G_1(u_0, u_1) \wedge G_2(u_1, u_2) \wedge \\ &\qquad\quad P_{Op,1}(i_0, i_1, u_0, u_1) \wedge P_{Op,2}(i_1, i_2, u_1, u_2)) \end{aligned} \tag{4.14}$$

$$\begin{aligned} &C_{Op,(1,2)}(u'_0, u'_2, o_0, o_2; i_0, i_2, u_0, u_2) \;= \\ &\qquad (\exists\, u'_1, o_1, u_1, i_1 \bullet \\ &\qquad\quad (G_1(u_0, u_1) \wedge C_{Op,2}(u'_1, u'_2, o_1, o_2; \ldots)) \vee \\ &\qquad\quad (C_{Op,1}(u'_0, u'_1, o_0, o_1; \ldots) \wedge G_2(u_1, u_2)) \vee \\ &\qquad\quad (C_{Op,1}(u'_0, u'_1, o_0, o_1; \ldots) \wedge C_{Op,2}(u'_1, u'_2, o_1, o_2; \ldots))) \end{aligned} \tag{4.15}$$

Proof. To show we have a retrenchment, we must show that the POs for the composed retrenchment follow from the POs for the individual ones: the initialisation PO, $Init(u'_2) \Rightarrow (\exists\, u'_0 \bullet Init(u'_0) \wedge G_{(1,2)}(u'_0, u'_2))$, follows immediately by composing the individual initialisation POs.

For the operation PO, let us assume (4.13), (4.14), and a step u_2 -(i_2, Op_0, o_2)-› u'_2 for some operation of Sys_2 whose name Op_0 is in Ops_0. Since (4.13) and (4.14) imply the existence of u_1, i_1 such that $G_2(u_1, u_2) \wedge P_{Op,2}(i_1, i_2, u_1, u_2)$ holds, from the second retrenchment we deduce that there are u'_1, o_1 such that there is a step of Sys_1, u_1 -(i_1, Op_0, o_1)-› u'_1, for which $G_2(u_1, u_2) \vee C_{Op,2}(u'_1, u'_2, o_1, o_2; \ldots)$ holds. Using the u_0, i_0 from (4.13), (4.14), and the step u_1 -(i_1, Op_0, o_1)-› u'_1, we repeat the argument to deduce the existence of u'_0, o_0 and a step u_0 -(i_0, Op_0, o_0)-› u'_0 for which $G_1(u_0, u_1) \vee C_{Op,1}(u'_0, u'_1, o_0, o_1; \ldots)$ holds. Therefore from (4.13), (4.14), and the step u_2 -(i_2, Op_0, o_2)-› u'_2, we have deduced a step u_0 -(i_0, Op_0, o_0)-› u'_0 such that $(\exists\, u'_1, o_1, u_1, i_1 \bullet (G_1 \vee C_{Op,1}) \wedge (G_1 \vee C_{Op,1}))$ holds. A little boolean algebra rearranges $(G_1 \vee C_{Op,1}) \wedge (G_1 \vee C_{Op,1})$ into $(\underline{G}_{(1,2)} \vee \underline{C}_{Op,(1,2)})$, where $\underline{G}_{(1,2)}$ and $\underline{C}_{Op,(1,2)}$ are $G_{(1,2)}$ and $C_{Op,(1,2)}$ with the existential quantifications removed. And since $(\exists \ldots \bullet (A \vee B)) \Rightarrow (\exists \ldots \bullet A) \vee (\exists \ldots \bullet B)$, we deduce that $(\exists \ldots \bullet (\underline{G}_{(1,2)} \vee \underline{C}_{Op,(1,2)}))$ implies $(G_{(1,2)} \vee C_{Op,(1,2)})$, giving the composed retrenchment operation PO. ☺

The above allows us to define the composition of the two retrenchments as being given by (4.13), (4.14), (4.15). A little more boolean algebra shows:

Proposition 4.4.2 The vertical composition of retrenchments given by (4.13), (4.14), (4.15) is associative.

The observant reader will have noticed that the roles played above by the structure of the composition argument on the one hand, and the algebraic structure of the $G \vee C$ term on the other, are essentially independent. This suggests that the same ideas will work for more elaborate notions of retrenchment. In [Banach and Poppleton (1999b)] the authors explore sharp retrenchment, where from the same hypotheses as here, instead of a pair of steps establishing $G \vee C$, they establish $(G \vee C) \wedge V$, where V is the nevertheless relation (and features the same variables as C). This form allows us for example to separate straightforward I/O remapping concerns which we would expect to hold globally, from the local failure of the retrieve relation for some steps. An associative law of vertical composition holds for this form, being given by (4.13), (4.14), (4.15), and (4.16).

$$\begin{aligned} &V_{Op,(1,2)}(u'_0, u'_2, o_0, o_2; i_0, i_2, u_0, u_2) \;= \\ &\qquad (\exists\, u'_1, o_1, u_1, i_1 \bullet \\ &\qquad\quad (V_{Op,1}(u'_0, u'_1, o_0, o_1; \ldots) \wedge V_{Op,2}(u'_1, u'_2, o_1, o_2; \ldots))) \end{aligned} \tag{4.16}$$

Pursuing these ideas further leads us to the idea of general retrenchment, wherein the initialisation PO is as before, but the operation PO reads:

$$\begin{aligned} &\Phi_{Op}(i, j, u, v) \wedge stp_{Op_C}(v, j, v', p) \;\Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_A}(u, i, u', o) \wedge \Theta_{Op}(u', v', o, p; i, j, u, v)) \end{aligned} \tag{4.17}$$

where Φ_{Op} and Θ_{Op} are formulae with a fairly general structure but containing a part that can be identified with the retrieve relation G. This is explored in [Banach (2001)] where it is shown that an associative law of vertical composition holds under suitable assumptions. General retrenchment enables us to move away from the 'one size fits all' formulation of retrenchment presented thus far, and to design bespoke retrench-

ment notions more precisely adapted to specific applications where the situation warrants it. Yet another direction in which the ideas of retrenchment may be generalised is exemplified by evolving retrenchment [Poppleton and Banach (2000)] in which the various clauses may depend on external parameters; the details of composition then depends on how these parameters are used. In this paper we continue to discuss only 'plain vanilla' retrenchment.

4.5 Previous Work

The authors are certainly not the first ones to raise concerns about the restrictiveness of refinement, and a number of extant works are related to the ideas in this paper. For instance, an early mention of the de facto reverse engineering of abstract levels from concrete ones is [Swartout and Balzer (1982)], while our distinguishing the world above the contracted model form the one below is related to the 'open-closed principle' of [Meyer (1988)].

The use of additional predicates during the refinement process appears in the rely/guarantee method of [Jones (1983)] and its successors. This also uses two predicates per operation, except that the consequent is conjunctive not disjunctive as in retrenchment. More directly comparable is the work on clean termination [Coleman and Hughes (1979), Blikle (1981)], dealing with the discrepancy between finite hardware oriented semantic domains and infinite idealised ones using proof theoretic techniques. This addresses one of the issues that makes retrenchment useful. Related to these papers is the thesis of Neilson [Neilson (1990)], which tackles the same problem by observing that the infinite idealised domains usually arise as convenient limits of the finite ones, and thus the idealised version of refinement arises as the limit of a finite version. This line of investigation leads to the notion of acceptably inadequate designs, and an interchange in the order of two quantifiers takes us from idealised refinement to finite refinement. Around the same time [Owe (1985), Owe (1993)] proposed program development using partial functions and a particularly convenient logic, prompted by finiteness and definedness considerations.

Shifting focus a little, retrenchment's ability to mix I/O and state between levels of abstraction, and specifically to change I/O representation, was anticipated in a refinement context in [Hayes and Sanders (1995)]. Of course this only makes sense when refinement is being used in a glass box manner; below the contracted model one must forbid change of I/O representation, which would otherwise be observable. More recently [Boiten and Derrick (1998a), Stepney, Cooper and Woodcock (1998), Mikhajlova and Sekerinski (1997)] also discuss refinement incorporating changes of I/O representation, while [Boiten and Derrick (1998b)] discuss grey box refinement, voicing concerns related to ours.

The need to occasionally violate a previously postulated retrieve relation, beyond simply coping with the dissonance between finite and infinite domains, has been addressed in a couple of works. In [Liu (1997)], the concept of evolution addresses this by demanding that the pre- and post- conditions of the abstract specification in a development step be semantically equivalent to subformulae of those at the more concrete level. Unfortunately since $((P \wedge Q) \vee (P \wedge \neg Q))$ is equivalent to P for any Q, the 'is a subformula of something equivalent to' property enables us to go from any Q to any P uncritically, tending to rob the evolution notion of semantic bite. Of course the same can be said to apply to retrenchment if one makes the within and con-

cedes relations strong enough; but in retrenchment, because these relations form part of the definition of the retrenchment step, in making these clauses strong, one is being explicit about just how extreme the relationship between the models is.

By contrast, in [Smith (2000), Smith and Fidge (2000)], building on the work of [Mahony (1992)], the concept of realisation approaches the same challenge by effectively permitting the substitution of variables in a specification by fresh ones satisfying new properties. Compared to retrenchment, one wins a little and loses a little thereby. One wins, since one can discover the fate of any property Π satisfied by the original specification, simply by applying the substitution to Π. One also loses however, since substitution, being a syntactic mechanism, implies that both old and new specifications are statements in the same language, and thus must have models within the same family of domains, limiting expressivity to some extent. With this proviso one can go from any system to any other, as with evolution, the leap being quantified by the substitution in question. Retrenchment, being rooted in the semantic arena, does not have the limitation indicated — see Section 6.2 for an example where the abstract and concrete domains are quite different — but the price for this is that tracking properties between the models becomes entirely nontrivial.

5 Initiation, Termination, and Simulation

In this section we examine the initiation and termination aspects of the operations of a system, and the way that these relate to retrenchment. The policy on initiation and termination in a program development formalism has a major impact on its internal structure, and differences in such policies are responsible in large part for the detailed differences between formalisms. The notion of correctness appropriate to a given formalism is invariably reflected in its initiation and termination policy.

5.1 Initiation and Termination

The simplest policy of all is to have no policy, and this has been our policy up to now. Because we wanted to illustrate retrenchment in the most transparent manner possible, we used pure transition systems, which typically come without any concept of a step starting but being unable to complete, or wanting to start but being unable to, or worse, being unable to start but completing successfully (so-called miracles). (Extreme possibilities such as the latter, typically emerge from the necessity to amplify the simple formalism of proper system states when accomodating nontermination. Since the logics used to reason about nontermination are insensitive to causal issues, the formulae that express nontermination for after-states have, by symmetry, duals for before-states, and these beg an interpretation.)

Beyond pure transition systems, most of the possibilities examined in the literature regarding what can happen as a result of trying to invoke an operation Op, can be covered by taking into account some criterion for its non-abortion or successful termination, $trm(Op)$, and of some further criterion for its successful initiation or feasibility, $fis(Op)$, also often called the guard. Although there is much consensus about how *trm* and *fis* behave mathematically, it is also the case that individual works take different and at times quite contradictory stances on what assumptions are appropriate for them. These range from assumptions concerning the significance of the individual notions with respect to the pragmatic understanding of what it means to invoke an operation within its environment, to assumptions concerning the relationship between

the two notions within the context of an individual system, and extend to the interplay between the two notions and refinement.

For instance [de Roever and Engelhardt (1998)] cover the most commonly encountered positions on the partial and total correctness frameworks from a contemporary perspective. Earlier, [Nelson (1989)] gave an accessible survey of a variety of approaches before detailing his own version of events. We also mention [Dijkstra and Scholten (1990), Hehner (1993), Abrial (1996a), Hoare and Jifeng (1998), Back and von Wright (1998)] which describe program development methodologies of this genre, and we recall that the more industrially targetted Z/VDM/RAISE techniques [Spivey (1993), Woodcock and Davies (1996), Jones (1990), Nielsen et al. (1989)] also have their perspectives on these issues. Of course the preceding citations are by no means exhaustive. Among other works which are relevant we mention [Doornbos (1994), Strulo (1996), Derrick et al. (1996), Dunne et al. (1998), Morris and Bunkenburg (1998), Morris and Bunkenburg (2000), Miarka et al. (2000)].

Although not universally true, in most of the formalisms mentioned, whenever both the *trm* and *fis* criteria are present, a refinement step has the potential to weaken the *trm* criterion and to strengthen the *fis* criterion. In fact one can often detect the impulse in a given framework, to name the *trm* criterion as that property connected with the successful accomplishment of an operation which can be weakened during refinement, and to name the *fis* criterion as that property connected with the successful accomplishment of an operation which can be strengthened during refinement. This, combined with earlier observations, can lead in the hands of some to what we might exuberantly call:

The Cynical Refinement Practitioner's Manifesto:

(1) Reverse engineer the abstract level from the concrete one.

(2) If during the refinement of an operation *Op* of the system under construction, it seems desirable to weaken some property Π connected with the successful accomplishment of *Op*, place Π in the *trm*(*Op*) predicate.

(3) If during the refinement of an operation *Op* of the system under construction, it seems desirable to strengthen some property Π connected with the successful accomplishment of *Op*, place Π in the *fis*(*Op*) predicate.

Software in the real world is overwhelmingly concerned with routines that initiate and terminate correctly. Consider for instance that the jet engines propelling a commercial airliner at 10,000 metres altitude contain quite a number of software artifacts that translate the pilot's high level commands into low level control directives for the engine components. How many of these potentially don't initiate or terminate correctly; how many ought to have the potential for incorrect initiation or termination?

By contrast, the diversity of theoretical views in the area of initiation or termination is overwhelmingly concerned with the distinctions that can be drawn amongst ways that routines might *fail* to initiate and/or terminate correctly. There is little disagreement about 'normal' execution steps.

Furthermore we make the observation that broadly similar degrees of success have been achieved for real world projects undertaken using a wide variety of formal techniques; techniques that differ substantially in their initiation and termination policy.

Were the details of initiation and termination policy decisive to the success of developments, this might not be so. All of which encourages a lightweight approach to matters of initiation and termination. Among work which takes such a view is the ASM work we cited earlier; also [Brinksma et al. (1987), Brinksma (1988)] take a transition system based approach to refinement which is comparatively lightweight.

Nevertheless the above does not mean that initiation and termination policy ought to be ignored. A question that gets to the heart of the matter is whether or not WHILE is to be considered an acceptable system constructor. If one decides to use a broad spectrum formal approach, in which system definition at all levels of abstraction is done within the same language, then since WHILE is inescapable at the lowest levels, it affects the highest levels too. Its potential for introducing nontermination and the attendant theoretical complexities, even when applied to finitary constituents, must be confronted at all levels. On the other hand if we benefit from the presence of the contracted model which plays a distinguished role in the development hierarchy, we can relegate looping constructs to a below-contracted-model implementation role, and keep the theory for specification constructors simpler. Our task in this paper is not to recommend one option above the other, but to recognise that both views may be legitimately held, and that therefore retrenchment must be able to confront a variety of approaches on the matter.

5.2 Partial Orders, Retrenchment and Stepwise Simulation

The preceding comments form a backdrop to the discussion of initiation and termination vis a vis retrenchment. One possibility would be to embark on a list of separate definitions of retrenchment, one for each variant of initiation and termination policy. This would be tedious, unbearably so considering the often small differences that exist between the different variants. Instead, we set out general principles according to which such definitions can be straightforwardly constructed for any variant that we might choose, yielding a 'brew your own retrenchment theory' capability.

One thing that all the approaches known to the authors have in common is that, regardless of how refinement is defined, and regardless of how *trm* and *fis* criteria figure in the refinement POs, refinement forms a partial order on systems. From this, and the observation that *trm* is weakened while *fis* is strengthened and that both are just predicates in the system variables, we conclude that the partial order property will hold regardless of whether we strengthen or weaken any given such criterion during a development step. This gives us a priori the freedom of either choice in retrenchment without endangering the partial order property. Now we can postulate the manner in which retrenchment should address initiation and termination policy (and any other issues concerned with the definition of correctness in force for any particular formalism). This should be according to the following:

Correctness in Retrenchment Formulation Principle:

> Any property (such as *trm* and *fis*) involved in the notion of correctness of an individual step, should in retrenchment, be *strengthened* in the passage from abstract to concrete system. This should be manifest in the structure of the retrenchment operation PO. (5.1)

For example, assuming we are dealing with a framework which features both *trm* and *fis* criteria, the operation PO (4.2) would be amplified to:

$$G(u, v) \wedge P_{Op}(i, j, u, v) \wedge trm_{Op_C}(v, j) \wedge fis_{Op_C}(v, j) \wedge stp_{Op_C}(v, j, v', p) \Rightarrow$$
$$trm_{Op_A}(u, i) \wedge fis_{Op_A}(u, i) \wedge$$
$$(\exists\, u', o \bullet stp_{Op_A}(u, i, u', o) \wedge (G(u', v') \vee C_{Op}(u', v', o, p; i, j, u, v))) \quad (5.2)$$

Here we have assumed that *trm* and *fis* are predicates in just the before-values though this is not necessary. Also the existence of a step like $stp_{Op_C}(v, j, v', p)$ is normally enough to guarantee the corresponding *fis* condition, but we have not bothered to assume that either. The fact that (5.2) generates a partial order is an easy extension of the compositionality of Section 4.4. More generally, the directive (5.1) would cause us to assemble a number of formulae of the form $\theta^i{}_C \Rightarrow \theta^i{}_A$, one for each property θ^i pertaining to correctness in the formalism in question. The overall retrenchment operation PO would then resemble:

$$(\bigwedge_i \theta^i{}_C) \wedge G \wedge P_{Op} \wedge stp_{Op_C} \Rightarrow$$
$$(\bigwedge_i \theta^i{}_A) \wedge (\exists\, u' \ldots \bullet stp_{Op_A} \wedge (G' \vee C_{Op})) \quad (5.3)$$

To give a specific example of (5.2), we quote the operation PO for retrenchment within the B-Method:

$$PA_{M,N} \wedge (I(u) \wedge G(u, v) \wedge J(v)) \wedge (\mathrm{trm}(T(v, j, p)) \wedge P(i, j, u, v, A)) \Rightarrow$$
$$\mathrm{trm}(S(u, i, o)) \wedge$$
$$[\ T(v, j, p)\ ] \neg [\ S(u, i, o)\ ] \neg (G(u, v) \vee C(u, v, o, p, A)) \quad (5.4)$$

In (5.4), $PA_{M,N}$ is the static context of abstract and concrete machines, $I(u)$ and $J(v)$ are the abstract and concrete invariants $G(u, v)$ being the retrieve relation, S and T are the generalised substitutions that define the abstract and concrete operations, and A is a logical variable introduced to enable before-values to be mentioned in the context of the after-valuation of the state variables (so $P(i, j, u, v, A)$ contains definitions of A and $C(u, v, o, p, A)$ uses of A). The $[T]\neg[S]\neg(G \vee C)$ structure is the B notation's predicate transformer version of our general $\forall Abs\text{-}Op \exists Conc\text{-}Op(G \vee C)$ form, and the right implication between feasibility criteria emerges naturally from (5.4) given the B feasibility definition. Incidentally, (5.4) is the appropriate definition regardless of whether we adopt standard B total correctness [Abrial (1996a)] or the general correctness perspective of [Dunne et al. (1998)].

It is worth pointing out that with the adoption of total correctness criteria, retrenchment has more not fewer cases to address. For example, the system (3.1) with the retrieve relation (2.5) and obvious initialisations, which was a refinement of (2.3) in our simple transition system framework, fails to be a refinement of (2.3) when we demand that trm_{Op_A} implies trm_{Op_C} in the refinement PO. Specifically, assuming that $trm_{Op_A}(u, i)$ holds iff there is an abstract *Op* transition starting from (u, i) and similarly for the concrete case, we observe that if $|\, mset\, | = 10$, then $trm_{put_A}(mset, i)$ is true since we can always add an element to *mset*, but assuming the retrieve relation $(\mathrm{mrng}(mseq) = mset)$, then $trm_{put_C}(mseq, i)$ is false as there are no put_C transitions starting from *mseq* when $\mathrm{length}(mseq) = 10$. Thus total correctness criteria for refinement increase the mutual interdependency between abstract and concrete models, making the construction of genuinely simpler abstract models more problematic. (Of course such a state of affairs is entirely justifiable *below* the contracted model, where a guarantee of conformance to the abstract model has to be provided, but is more questionable above it.)

The impetus in retrenchment is to alleviate the interdependency tendency, and the orientation of the dependency between trm_{Op_A} and trm_{Op_C} in the retrenchment PO is designed with this in mind. The advantages of the scheme in (5.1) and (5.2) are twofold. Firstly all the dependencies flow unidirectionally, which makes it easier to have features in one of the models (the abstract one), that are ignored completely in the other. Secondly it makes it easier, in principle at least, to prove stepwise simulation results.

To substantiate these claims we initially explore their consequences in the simpler world of refinement and with I/O absent. Taking it for granted that any execution step implies its own *fis*, *normal* refinement can be briefly expressed by the operation PO:

$$G \wedge trm_{Op_A} \wedge stp_{Op_C} \;\Rightarrow\; trm_{Op_C} \wedge (\exists \ldots \bullet stp_{Op_A} \wedge G') \tag{5.5}$$

Inverting the *trm* dependency, we get *inverted* refinement, expressed by the PO:

$$G \wedge trm_{Op_C} \wedge stp_{Op_C} \;\Rightarrow\; trm_{Op_A} \wedge (\exists \ldots \bullet stp_{Op_A} \wedge G') \tag{5.6}$$

Consider now proving a result like Proposition 2.3.1. We assume given a concrete execution sequence $\mathcal{T} = [\ v_0 \text{-}(Op_{C,0})\text{-}\!\rightarrow v_1 \text{-}(Op_{C,1})\text{-}\!\rightarrow v_2 \ldots v_n\ ]$ and we want to inductively construct an abstract execution sequence $\mathcal{S} = [\ u_0 \text{-}(Op_{A,0})\text{-}\!\rightarrow u_1 \text{-}(Op_{A,1})\text{-}\!\rightarrow u_2 \ldots u_n\ ]$ satisfying $(\forall\, r \bullet G_r)$ which is the simplified form of (2.10). Assuming that an execution step implies its own *trm* property[3], (5.6) yields the required induction for the inverted case effortlessly. Just as in Proposition 2.3.1, because $stp_{Op_C} \Rightarrow trm_{Op_C}$, each concrete step by its very existence yields all the antecedents required in (5.6). The normal case is quite different however. Since $stp_{Op_C} \not\Rightarrow trm_{Op_A}$, we need additional hypotheses to perform the induction. What is required is a condition such as:

$$wp_A(G_0 \wedge \ldots \wedge wp_A(G_{n-1} \wedge wp_A(G_n))\ldots) \neq \mathsf{false} \tag{5.7}$$

where wp_A is the weakest precondition predicate transformer in the abstract system, and where the free values of concrete state variables occurring in the G_r are regarded as parameters in (5.7). Not only is this an unwieldy condition to assume, it is not stable as the length of $\mathcal{T}$ increases to infinity, unlike the inverted refinement case where the length of $\mathcal{T}$ is immaterial to the proof. In the normal refinement case, for infinite $\mathcal{T}$, we need different conditions and a different proof strategy based on König's Lemma. (The required argument appeared in [Abadi and Lamport (1991)], see also [Jonsson (1989), Jonsson (1991)]. For its application in detail to stepwise simulation in the context of the B-Method and retrenchment see [Banach and Poppleton (1999b), Banach and Poppleton (2000a)].)

Note that in contrast to remarks above, in the inverted orientation of refinement expressed by a PO like (5.6), (3.1) remains a refinement of (2.3) despite taking the *trm* criterion into account.

The contrast between normal and inverted refinement is illustrated vividly by a game theoretic reformulation of the inductive proof strategy for stepwise simulation:

Stepwise Simulation Game for Refinement:

C_{2n}: Play $v_{r+1} \bullet stp_{Op_{C,r}}(v_r, v_{r+1}) \wedge G(u_r, v_r)$

A_{2n+1}: Reply $u_{r+1} \bullet stp_{Op_{A,r}}(u_r, u_{r+1}) \wedge G(u_{r+1}, v_{r+1})$ (5.8)

3. It is not unreasonable to identify the execution steps of a system with those elements of its transition relation satisfying both *trm* and *fis*.

The simulation succeeds if player A has a winning strategy. This is guaranteed for inverted refinement, since for each concrete step player C plays, $stp_{Op_C} \Rightarrow trm_{Op_C}$ allowing application of the PO, and hence a player A reply. For normal refinement, the PO alone does not guarantee a winning strategy, since when player C plays, he does not fulfil the PO's premises; therefore we need additional ammunition such as (5.7).

We now turn to the situation for retrenchment, where the stepwise simulation game is a little different. With I/O restored it reads:

Stepwise Simulation Game for Retrenchment:

C_{2n}: Play $v_{r+1}, j_r, p_{r+1} \bullet stp_{Op_{C,r}}(v_r, j_r, v_{r+1}, p_{r+1}) \wedge$
$G(u_r, v_r) \wedge P_{Op_r}(i_r, j_r, u_r, v_r)$

A_{2n+1}: Reply $u_{r+1}, i_r, o_{r+1} \bullet stp_{Op_{A,r}}(u_r, i_r, u_{r+1}, o_{r+1}) \wedge$
$(G(u_{r+1}, v_{r+1}) \vee C_{Op_r}(u_{r+1}, v_{r+1}, o_{r+1}, p_{r+1}; i_r, j_r, u_r, v_r))$ (5.9)

Again the simulation succeeds if player A has a winning strategy. This time, although we can once more use $stp_{Op_C} \Rightarrow trm_{Op_C}$ to get part of the way towards exploiting the operation PO, nevertheless player C is not always guaranteed a move, despite having the initiative, because he has to assert $G(u_r, v_r) \wedge P_{Op}(i_r, j_r, u_r, v_r)$, whereas opponent A has only established $G(u_r, v_r) \vee C_{Op}(u_r, v_r, o_r, p_r; \ldots)$ in the previous round. The gap between these, illustrates why in retrenchment we need additional properties as discussed in Section 4.3 to establish stepwise simulation.

There is an important difference between the normal refinement and retrenchment game scenarios, in that the character of the additional hypotheses needed to assure a player A winning strategy tends to be different in the two cases. For normal refinement, a player A reply has to not only yield $stp_{Op_{A,r}}(u_r, u_{r+1}) \wedge G(u_{r+1}, v_{r+1})$ for the current round, but also to tacitly assure $trm_{Op_{A,r+1}}(u_{r+1})$ for the next round, if the operation PO is going to be used profitably in the next round. Thus properties of adjacent steps are combined, making an assumption like (5.7) a global one, depending in detail on all the steps of the concrete sequence being simulated. In contrast, as a result of the inverted orientation of the *trm* dependencies in retrenchment, only the gap between the $G \vee C_{Op_r}$ established in one round and the $G \wedge P_{Op_{r+1}}$ needed in the next has to be bridged. Now both the $G \vee C_{Op_r}$ and the $G \wedge P_{Op_{r+1}}$ refer predominantly to the same states. Thus in those cases in which the concedes relations do not refer to the before-values, we have a decoupling of the assumptions needed at different points in the state space, making the assumptions needed local, rather than global. Thus an important rationale for principle (5.1) is that it helps to promote the derivation of stepwise simulation properties that are facilitated by local rather than global additional assumptions.

When a stepwise simulation result can be derived assisted by purely local assumptions, it often feeds directly through into a strong simulation result, because essentially the same assumptions will do at both the beginning and end of a transition, assisting the construction of the relations Θ_S and Θ_L of the strong simulation in a purely local manner. When the assumptions needed are not local in this sense, history information usually needs to be encoded into the strong simulation state space, wrecking the smooth correspondence. Thus a further rationale for principle (5.1) is that it helps to assure that the strong simulation results arising from stepwise simulations do so smoothly, without needing extraneous history information. See once more [Banach and Poppleton (1999b), Banach and Poppleton (2000a)] for specific details.

The preceding helps to illuminate an important difference in emphasis between refinement and retrenchment regarding sequential composition. In refinement, the major thing is monotonicity of all constructs with respect to the refinement partial order, including sequential composition. For a calculational approach to implementation below the contracted model, based on the strict preservation of the properties captured in the system invariants and retrieve relation, this is entirely appropriate. In retrenchment, where strict adherence to the retrieve relation is de facto not demanded, and which allows a looser coupling between abstract and concrete systems, the major criterion is stepwise simulation, and when it is tied to the PO principle (5.1), it is more benign with respect to sequential composition.

One byproduct of these observations is that the complex nature (and instability for infinite sequences) of the assumption (5.7) needed to establish stepwise simulation for normal refinement, is not as surprising at it might at first appear. When one places the emphasis on monotonicity of sequential composition with respect to the refinement partial order, the *trm* criterion for a composite emerges as an *output* of the calculation. Also the focus of attention in the hypotheses of the relevant theorem is on the midpoint, where the two component refinements meet, and this distracts one's gaze away from what is going on at the beginning of the sequence. When one places the emphasis on stepwise simulation, the relevant property is required as an *input* to the calculation, where its complex nature as a hypothesis is more jarring. Also the focus of attention in the theorem is right there, on the hypotheses, amplifying this effect; see [Poppleton (2001), Poppleton and Banach (1999)] for more details. The whole phenomenon can be understood as a permutation of meta level quantifiers between the pertinent theorems.

6 Case Studies

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. Regarding the latter, we discuss one focused example in reasonable detail, and another from a wider architectural perspective.

6.1 Modelling the Real World

As the application of formal methods for system development in the real world continues to grow, so does the interest in applying them to systems which capture the properties of physical situations requiring continuous mathematics for their description. However traditionally understood formal methods are largely inappropriate for the task. The trouble is that continuous mathematics lies at quite a distance from the discrete finitistic logic-based formalisms that formal methods rely on. Which is not to say that one cannot build models of portions of continuous mathematics within logical and algebraic foundations — indeed there are many contemporary research directions that may be understood as assisting precisely such an objective (at least as a side effect), eg. formal topology [Sambin (1987), Sambin and Gebellato (1999)], computable reals [Bajard et al. (2000)], computable analysis [Weihrauch (2000)], etc., quite aside from the classical constructions of complexes via reals, rationals, integers, naturals à la Weierstrass and Cantor. However the complexity of these undertakings denies their efficient adoption as a formal basis for applied mathematics in engineering applications; not to mention the fact that variants of these are subtly in-

compatible regarding what can be expressed or deduced within them, giving rise to a Babelisation of 'formalisations of continuous mathematics' which is hardly useful in real engineering applications.

Still, the fact that continuous mathematics has been done with rigour for a century or more is evidence that what has been done ought to be capable of being captured formally. The relative 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 (though arguably, the situation is slowly improving). In fact basic aspects of analysis have been examined from the mechanical theorem proving point of view [Harrison (1998)], and this work shows that a formal approach is entirely feasible, but is no less of a job than indicated above. And this is yet different to what may be found in 'computer algebra' systems such as Maple and Mathematica, with their more ad hoc foundations.

Aside from the complexity issue, is the scale issue. The sheer breadth of continuous 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. Equally unrealistic is reinventing the core continuous mathematics wheel formally as a precursor to the more specialised reasoning required in a specific application. Not only is the cost of deriving any significant piece of applied mathematics from first principles prohibitive, but many different developments would overlap significantly in the mathematics needed, 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 formally support the specifics of the development. Just 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 formal development technology, would become a matter for engineering judgement.

The suggested approach via axioms at a suitable level of abstraction, could be extended to incorporate the heuristic and semi-empirical reasoning that often forms an indispensable part of real world engineering methodology. Suppose for example that it is believed that within certain bounds of applicability, such and such an 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 complex system of equations. Then that belief can be expressed as a rule of the formal development framework, and its use controlled by the same reasoning technology that supports the rest of the formal development. In fact this axiomatic approach is in many ways reminiscent of computer algebra, aside from the fact that to certifiably underpin critical developments, the axiomatics and reasoning would have to be open to scrutiny, rather than being hidden inside a commercially protected binary object.

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 obviously these questions are of vital importance if the discrete model is 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 formal justification of the appropriate development steps in real world applications where appropriate. Evidently incorporating them into strict refinement steps is too much to ask in general, and the greater flexibility of the retrenchment formalism is much better suited to the task in hand, as we now show.

6.2 Continuous and Digital Control, and Retrenchment

Most applications of digital computers to physical problems involve control of a physical plant. In such work the continuous component is time, and the problem is to describe and control a one dimensional dynamics typically governed by differential equations relating derivatives of controlled variables to their values and the values of external inputs etc. In addition to the typically smooth evolution of the system, it may be subjected from time to time to certain discrete events which disrupt the otherwise continuous behaviour. Over the years, a large amount of work has been directed at taming the difficulties that arise. See for example [Maler (1997), Alur, Henzinger and Sontag (1996)].

It turns out that the digital control paradigm is an ideal application for retrenchment. The design of such a system often starts at the physical level, done using continuous mathematics. Once the design of the continuous control has resulted in a system with the desired behaviour, thought may be given to the discrete control version. See eg. [Kuo (1992), Franklin et al. (1998)]. It is well known that the choice of discretisation scheme can affect the relationship between continuous and discrete systems in a manner which may be at times smooth, at times catastrophic. It is hopeless therefore to try to address the design space using refinement; so much ingenuity would have to go into choosing the abstract model wisely in order than a refinement relation might arise (assuming one might arise at all), that the point of ever having the abstract model would largely disappear. More importantly, each change of discretisation scheme that was considered, would entail the reengineering of the abstract model for reasons similar to ones discussed in Section 3.1. *Needing to redo higher levels of abstraction in the light of decisions that one can predict one will need to make later, and for which the delay in consideration is completely justifiable on engineering grounds, is bad news for a development methodology.* We will show that retrenchment does not suffer from this major drawback, allowing the abstract model to remain unaltered while the concrete model is changed, allowing the *reuse* of the abstract model in a scenario where refinement, assuming it was even feasible, would not.

We consider a control redesign situation, kept simple to prevent the scale of the enterprise from overwhelming the remainder of the paper. The problem, posed in Laplace transform space as is common in engineering situations, is given by:

$$sx_C(s) = A_C x_C(s) + B_C r_C(s) \tag{6.1}$$

where $x_C(s)$ is the Laplace transform of the continuous system state, A_C and B_C are constants, and $r_C(s)$ is the Laplace transform of a continuous external input signal.

Because retrenchment, as described in this paper, can only speak about single state transitions, we have to address the control problem in the state space formulation.[4] Accordingly, in real time, the continuous system is described by the differential equation:

$$\dot{x}_C(t) = A_C x_C(t) + B_C r_C(t) \tag{6.2}$$

where $\dot{x}_C(t)$ is the time derivative of $x_C(t)$.

In fact the slight digression into Laplace transforms is instructive. The entry of Laplace transforms into the fray brings with it implicitly the extremely strong properties of (at least piecewise) analyticity, and L_2 analysis. Engineering mathematics becomes dust without such assumptions properly applied. The routine engineering manipulations between space or time domains on the one hand, and real or complex frequency domains on the other become invalid unless informed by them. Recalling the foundational approaches to analysis mentioned in Section 6.1, we therefore see that unless they were to be carried through as far as the Cauchy-Riemann equations and their consequences, and the more well known facets of L_2 analysis, they would not really be in a position to support most engineering applications.

One possible discrete system corresponding to (6.2) is given by:

$$\Delta^{+T} x_D(k) = A_D x_D(k) + B_D r_D(k) \tag{6.3}$$

where x_D is the discrete system state, $\Delta^{+T} x_D(k)$ is its forward difference for sampling period T, given by:

$$\Delta^{+T} x_D(k) = \frac{x_D(k+1) - x_D(k)}{T} \tag{6.4}$$

and A_D and B_D are constants, r_D being the discrete external input signal. The solution of (6.3) for the next state is immediate:

$$x_D(k+1) = (1 + TA_D)x_D(k) + TB_D r_D(k) \tag{6.5}$$

while the solution of (6.2) is standard, and for a period T to the future of a starting point $t = kT$, is given by:

$$x_C((k+1)T) = e^{A_C T} x_C(kT) + \int_0^T e^{A_C(T-\tau)} B_C r_C(kT + \tau)\, d\tau \tag{6.6}$$

4. The extension of retrenchment to encompass more global aspects of execution sequences holds out the prospect of incorporating the various frequency domain based approaches to control engineering (including Laplace transforms) into the retrenchment formalism. This however is outside the scope of this paper.

To obtain a simple comparison with the discrete case, we expand the exponentials into their power series, and expand $r_C(kT + \tau)$ using Taylor's Theorem before integrating term by term. Keeping only terms up to $O(T)$:

$$x_C((k+1)T) = (1 + TA_C)x_C(kT) + TB_C r_C(kT) + o(T) \tag{6.7}$$

so that:

$$x_C((k+1)T) - x_D(k+1) = (x_C(kT) - x_D(k)) + T(A_C x_C(kT) - A_D x_D(k)) + T(B_C r_C(kT) - B_D r_D(k)) + o(T) \tag{6.8}$$

Now at time $(k+1)T$, the difference between the two states, $x_C((k+1)T) - x_D(k+1)$, becomes comparable to that at time kT , i.e. $x_C(kT) - x_D(k)$, provided:

$$A_C x_C(kT) - A_D x_D(k) + B_C r_C(kT) - B_D r_D(k) = o(1) \tag{6.9}$$

This illustrates that by adjusting the external demand suitably, one can keep the continuous and discrete controls in line, an unsurprising proposition. Also this is based on an extremely simple approximation. If we retain second order terms in T then (6.5), being exact, remains unchanged, but (6.6) acquires more terms, which feed through to (6.8), and to (6.9) which becomes:

$$A_C x_C(kT) - A_D x_D(k) + B_C r_C(kT) - B_D r_D(k) + \frac{T}{2}(A_C{}^2 x_C(kT) + A_C B_C r_C(kT) + B_C \dot{r}_C(kT)) = o(T) \tag{6.10}$$

where $\dot{r}_C(kT)$ is the time derivative of r_C at kT. Higher order terms in T would bring the appearance of higher derivatives of r_C at kT of course. These indicate that the details of the shape of r_C between kT and $(k+1)T$ have an important bearing on the conformance between discrete and continuous systems. Detailed calculations are beyond the scope of this paper.

The facts we now have can be interpreted as retrenchments in the following manner. First we have to set up our transition systems. The abstract system is given by a transition relation for an abstract operation Op_A as follows:

$$(x_C(t), \dot{x}_C(t)) \text{ -}(r_C(t), Op_A)\text{-}\rightarrow (x_C(t'), \dot{x}_C(t')) \tag{6.11}$$

where t and t' are nonnegative real valued and $t < t'$; and there is a requirement that there exists a (global) solution of (6.2), such that for any $0 < t < t'$ the solution agrees with the quantities appearing in (6.11) at t and t'. Standard theory [Jeffreys and Jeffreys (1956), Hildebrand (1962)] says that there is enough information recorded in the state vector $(x_C(t), \dot{x}_C(t))$, to ensure that a unique analytic solution is determined by it and the input, on which (6.6) is based.

The concrete system is given by a transition relation for a concrete operation Op_C of the following kind:

$$x_D(k) \text{ -}(r_D(k), Op_C)\text{-}\rightarrow x_D(k+1) \tag{6.12}$$

where k is natural valued and the quantities appearing in (6.12) must constitute a solution of (6.5). Clearly there is enough information recorded in the state $x_D(k)$ to ensure that a unique solution to (6.5) is determined by it and the input.

As a simple retrieve relation we will take:

$$G(x_C(kT), x_D(k)) = | x_C(kT) - x_D(k) | \leq \varepsilon \tag{6.13}$$

where ε is sufficiently small (or indeed even zero). Now (6.9) lends itself to a retrenchment reinterpretation with within and concedes relations:

$$P_1(r_C(kT), r_D(k), x_C(kT), x_D(k)) = \\ | A_C x_C(kT) - A_D x_D(k) + B_C r_C(kT) - B_D r_D(k) | \leq o(1) \tag{6.14}$$

$$C_1(x_C((k+1)T), x_D(k+1); r_C(kT), r_D(k), x_C(kT), x_D(k)) = \\ | x_C((k+1)T) - x_D(k+1) | \leq o(1) \tag{6.15}$$

If we wish to extend the the argument to second order we may do so, though we would have to include $\dot{r}_C(kT)$ as an additional input in the formulation of (6.11) because of its presence in (6.10). Assuming this, (6.10) yields:

$$P_2(r_C(kT), r_D(k), x_C(kT), x_D(k)) = \\ | A_C x_C(kT) - A_D x_D(k) + B_C r_C(kT) - B_D r_D(k) + \\ + \frac{T}{2} (A_C{}^2 x_C(kT) + A_C B_C r_C(kT) + B_C \dot{r}_C(kT)) | \leq o(T) \tag{6.16}$$

$$C_2(x_C((k+1)T), x_D(k+1); r_C(kT), r_D(k), x_C(kT), x_D(k)) = \\ | x_C((k+1)T) - x_D(k+1) | \leq o(T) \tag{6.17}$$

As retrenchments, these say that if the discrete system makes a step, then provided the appropriate conditions hold, the continuous system's behaviour will be acceptable. Note that we do not have absolute bounds on the errors involved due to the extreme simplicity of our analysis. We could have done a little better by using one of the exact forms of Taylor's Theorem, at the cost of some clutter; but the essential point is well enough made as it is.

More incisive results yet could be obtained, but only at the cost of much greater effort. This is due to the necessity of working with convolutions when we focus on the time domain as these are less easy to estimate in a straightforward way; it is also well known that if the demand input is sufficiently troublesome, the time domain behaviour of a (continuous) linear system may be prone to large local deviations, despite the fact that the overall system performance (say in the L_2 sense) may be perfectly acceptable. These facets explain why the majority of engineering analysis of systems resides in the frequency domain.

One aspect that is clear even in our elementary treatment, is that retrenchment has cleanly separated out all aspects that involve the sampling period T from the abstract model. They all reside in the retrenchment data and concrete model. Thus consideration of the sampling period may be postponed to an appropriate point, without worrying about the abstract model, which is appropriately simple and remains unchanged in the presence of the discrete model. The fact that this is possible is one of the major strengths of the retrenchment approach. This remains true even in a total correctness formulation where in refinement, one would be obliged to say something about all possible continuous behaviours (that share the same $x_C(kT)$, $r_C(kT)$ in our example) to establish the relevant PO. To avoid saying anything about clearly undesirable behaviours, would require a skilful construction of the abstract model, to ensure that they did not form part of it. In retrenchment this is not necessary — even in more precise analyses we would need to exhibit only one witnessing continuous behaviour

for any discrete behaviour; we might construct such a thing based on suitable interpolations of the discrete behaviour.

6.3 Radiotherapy Dose Calculations

Aside from control problems, for which approaches towards exact solutions exist even if chiefly for textbook examples, there are also many problems requiring software implementation that require applied mathematics of a different kind. A typical case in point 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 way that simple calculations with natural numbers are justified on the basis of the Peano axioms in a modern theorem prover.

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. This is the essence of specification, asserting 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, in typical finite element fashion. 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.

6.4 Summary

Surveying the above two examples, we see that much of the reasoning that would otherwise fall outside of the remit of formal development can potentially be brought into the fold by the use of retrenchment, at least in principle. 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 process also reveals the mathematically most challenging parts for what they are, and documents to what extent they have been resolved through utilising either rigorous results from analysis, or the adoption of pragmatic engineering intuition.

7 Conclusions

In the preceding sections we have surveyed the way the refinement has evolved in recent times to address not only the providing of guaranteed properties of specifications during the implementation of a contracted model, but also the desirability of acting as a specification contructor above it. In the latter arena, it sometimes proves restric-

tive regarding the relationships between models that it can express, and this provides the spur for the introduction of retrenchment. Being a weaker notion, retrenchment permits the incorporation of models within a development path that could not be brought together using refinement alone, and this gives developers more flexibility. It has to be stressed that retrenchment must be confined to the above-contracted-model parts of developments, where the incompatibilities between levels of abstraction that it permits are transparent and form part of the dialogue between user and supplier at system definition time. Since for large and complex systems, particularly those interfacing to continuous physical models, system definition is a lengthy and nontrivial process, and substantial parts of it are beyond the reach of refinement, retrenchment can help to smoothly integrate this early activity into a formal methodology in which refinement can ultimately take over to guarantee the properties of the contracted model.

Properly substantiating such a contention requires more thorough examination of a number of issues. For instance, the interplay between retrenchment and refinement needs to be properly explored to show that the two techniques can work together as smoothly as claimed, see [Banach (2000), Banach et al. (2001a)]. Also some larger scale case studies need to be described to demonstrate how retrenchment stands up in the face of realistic problems, see [Banach et al. (2001b)]. Another crucial question concerns the way system properties are transformed through retrenchment steps. Given the potential disparity between the two models in a retrenchment, this is of interest in both the upwards and downwards directions. The challenging nature of this issue is already witnessed by the relatively weak simulation properties of retrenchment that we discussed above, and is exacerbated by the possibilities for punctured simulation opened up by retrenchment. These questions will be explored elsewhere. On the positive side, the identification of stepwise simulation as central semantic notion in retrenchment, in contrast to monotonicity of composition as central semantic notion for refinement, gives a clear focus to such investigations. In this paper it has already clarified how correctness and related properties ought to transform under retrenchment, allowing the key principles of a 'brew your own retrenchment correctness theory' philosophy to be set out.

Acknowledgements

Since the initial introduction of retrenchment in [Banach and Poppleton (1998)], many peoples' comments have helped the authors to refine (rather than retrench) its presentation into what appears in this paper. The authors would particularly like to thank Tom Anderson, Didier Bert, Juan Bicarregui, Jean-Paul Bodeveix, Egon Börger, Eerke Boiten, Michael Butler, Trevor Cockram, John Derrick, Steve Dunne, Mamoun Filali, Andy Galloway, Jon Hall, Kung-Kiu Lau, Shaoying Liu, John McDermid, Dave Neilson, Ken Robinson, Graeme Smith, Ib Sorensen, Susan Stepney, Bill Stoddart, Sam Valentine, Steve Vickers, and a number of anonymous referees.

References

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

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

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

Andrews B. (1986); An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Academic.

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

Back R. J. R., Sere K. (1996); Superposition Refinement of Reactive Systems. Form. Asp. Comp. **8**, 324-346.

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.

Back R. J. R., von Wright J. (1998); Refinement Calculus, A Systematic Introduction. Springer.

Banach R. (2000); Maximally Abstract Retrenchments. *in*: Proc. IEEE ICFEM-00, 133-142.

Banach R. (2001); General Retrenchment. *work in progress*

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

Banach R., Poppleton M. (1999a); Retrenchment and Punctured Simulation. *in*: Proc. IFM-99, Araki, Gallway, Taguchi (eds.), 457-476, Springer.

Banach R., Poppleton M. (1999b); Sharp Retrenchment, Modulated Refinement and Punctured Simulation. Form. Asp. Comp. **11**, 498-540.

Banach R., Poppleton M. (2000a); Retrenchment, Refinement and Simulation. *in*: Proc. ZB-00, Bowen, Dunne, Galloway, King (eds.), LNCS **1878**, 304-323, Springer.

Banach R., Poppleton M. (2000b); Fragmented Retrenchment, Concurrency and Fairness. *in*: Proc. IEEE ICFEM-00, 143-151.

Banach et al. (2001a); Algebraic properties of Retrenchment and Refinement. *work in progress*

Banach et al. (2001b); Control Engineering: A Killer Application for Retrenchment. *work in progress*

Bajard J.-C., Frougny C., Kornerup P., Muller J.-M. (eds.) (2000); Proc. Fourth Conference on Real Numbers and Computers. Schloss Dagstuhl, Germany, Dagstuhl Conference No. 00162, DMTCS, Springer.

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

Börger E. (1995a); Why use Evolving Algebras for Hardware and Software Engineering? *in*: Proc. SOFSEM-95, Bartosek, Staudek, Wiedermann (eds.), LNCS **1012**, 236-271, Springer.

Börger E. (ed.) (1995b); Specification and Validation Methods. Oxford University Press.

Börger E. (1999); High Level System Analysis and Design using Abstract State Machines. *in*: Proc. FM-Trends-98, Hutter, Stephan, Traverso, Ullmann (eds.), LNCS **1641**, 1-43, Springer.

Börger E., Schulte W. (1998); A Programmer Friendly Modular Definition of the Semantics of Java. *in*: Formal Syntax and Semantics of Java, Alves-Foss (ed.), LNCS **1523**, 353-404, Springer.

Börger E., Schulte W. (2000); Modular Design for the Java Virtual Machine. *in*: Architecture Design and Validation Methods. Börger (ed.), 297-357, Springer.

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

Boiten E., Derrick J. (1998b); Grey Box Data Refinement. *in*: Proc. IRW&FMPacific-98, Grundy, Schwenke, Vickers (eds.), Disc. Maths. and Theor. Comp. Sci. 45-59, Springer.

Brinksma E., Scollo G., Steenbergen C. (1987); LOTOS Specifications, their Implementations and their Tests. *in*: Proc. IFIP Protocol Specification, Testing and Verification (PSTV-6), Sarikaya, Bochmann (eds.), 349-360, Elsevier (North-Holland).

Brinksma E (1988); A Theory for the Derivation of Tests. *in*: Proc. IFIP Protocol Specification, Testing and Verification (PSTV-8), Aggarwal, Sabnani (eds.), 63-74, Elsevier (North-Holland).

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

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

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

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

Dijkstra E. W. (1972); Notes on Structured Programming. *in*: Structured Programming, Academic Press.

Dijkstra E. W., Scholten C. S. (1990); Predicate Calculus and Program Semantics. Springer.

Doornbos H. (1994); A Relational Model of Programs without the Restriction to Egli-Milner Constructs. *in*: Proc. PROCOMET-94, Olderog (ed.), 357-376, IFIP.

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

Ehrig H., Grosse-Rhode M. (1994); Functorial Theory of Parameterised Specifications in Generalised Specification Frameworks. Theor. Comp. Sci. **135**, 221-266.

Ehrig H., Mahr B. (1985); Fundamentals of Algebraic Specification I. Springer.

Ehrig H., Mahr B. (1990); Fundamentals of Algebraic Specification II. Springer.

Fiadeiro and Maibaum (1997); Categorical Semantics of Parallel Program Design. Sci. Comp. Prog. **28**, 111-138.

Francez N., Forman I. R. (1990); Superimposition for Interactive Processes. in: Proc. CONCUR-90, Baeten, Klop (eds.), LNCS **458**, 230-245, Springer.

Franklin M. L., Powell G. F., Workman J. D. (1998); Digital Control of Dynamic Systems. Addison Wesley Longman.

Harrison J. R. (1998); Theorem Proving with the Real Numbers. Springer. *also* PhD. Thesis, Cambridge University Computing Laboratory 1996, *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.

Hehner E. C. R. (1993); A Practical Theory of Programming. Springer.

Hildebrand F. B. (1962); Advanced calculus for Applications. Prentice-Hall.

Hoare C. A. R. (1972); Proof of Correctness of Data representations. Acta Inf. **1**, 271-281.

Hoare C. A. R., Jifeng H. (1998); Unifying Theories of Programming. Prentice-Hall.

Hodges (1993); Model Theory. Cambridge University Press.

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.

Jeffreys H., Jeffreys B. S. (1956); Methods of mathematical Physics. 3rd ed., Cambridge University Press.

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

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

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

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

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

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

Katz S. (1993); A Superimposition Control Construct for Distributed Systems. ACM Trans. Prog. Lang. Sys. **15**, 337-356.

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

Kuo B. C. (1992); Digital Control Systems. (2nd ed.) Oxford University Press.

Lamport L. (1989); A Simple Approach to Specifying Concurrent Systems. Comm. ACM **32**, 32-45.

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

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

Liu S. (1997); Evolution: A More Practical Approach than Refinement for Software Development. *in*: Proc. ICECCS-97, 142-151, IEEE.

Mahony B. (1992); The Specification and Refinement of Timed Processes. PhD. Thesis, Dept. of Computer Science, Queensland University.

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

Manna Z., Pnueli A. (1992); The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer.

Meyer (1988); Object Oriented Construction. Prentice-Hall.

Miarka R., Boiten E., Derrick J. (2000); Guards, Preconditions and Refinement in Z. *in*: Proc. ZB-00, Bowen, Dunne, Galloway, King (eds.), LNCS **1878**, 286-303, Springer.

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

Morgan C. (1994); Programming from Specifications. 2nd ed., Prentice-Hall.

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

Morris J. M., Bunkenburg A. (1998); Partiality and Nondeterminacy in Program Proofs. Form. Asp. Comp. **10**, 76-96.

Morris J. M., Bunkenburg A. (2000); Term Transformer Semantics, ACM Trans. Prog. Lang. Sys. *to appear*.

Nielsen M., Havelund K., Wagner K. R., George C. (1989); The RAISE Language Method and Tools. Form. Asp. Comp. **1**, 85-114.

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

Nelson G. (1989); A Generalisation of Dijkstra's Calculus. ACM Trans. Prog. Lang. Sys. **11**, 517-561.

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

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

Poppleton M. R. (2001); Formal Methods for Continuous Systems: Liberalising Refinement in B. PhD. Thesis Manchester University Department of Computer Science.

Poppleton M. R., Banach R. (1999); Retrenchment: Extending the Reach of Refinement. *in*: Proc. IEEE ASE-99, 158-165.

Poppleton M. R., Banach R. (2000); Retrenchment: Extending Refinement for Continuous and Control Systems. *in*: Proc. IWFM-00, 26pp., `http://ewic.org.uk/ewic/workshop/view.cfm/IWFM-00`

Sambin G. (1987); Intuitionistic Formal Spaces — a First Communication. Mathematical Logic and its Applications, Skordev (ed.), 187-204, Plenum, New York.

Sambin G., Gebellato S. (1999); A Preview of the Basic Picture: A New Perspective on Formal Topology. *in*: Proc. TYPES-98, Altenkirch, Naraschewski, Reus (eds.), LNCS **1657**, 194-207, Springer.

Schneider F. B. (1997); On Concurrent Programming. Springer.

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

Smith G. (2000); Stepwise Development from Ideal Specifications. *in*: Proc. IEEE ACSC-00, (Aus. Comp. Sci. Comm. **22**), 227-233.

Smith G., Fidge C. (2000); Incremental Development of Real-Time Requirements: The Light Control Case Study. JUCS **6**, 704-730.

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

Srinivas Y. V., Jullig R. (1995); Specware(TM): Formal Support for Composing Software. Proc. MPC-95, Moller (ed.), LNCS **947**, 399-422, Springer. *also*: Kestrel Institute Technical Report KES.U.94.5. `http://www.kestrel.edu`

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

Strulo B. (1996); How Firing Conditions Help Inheritance. *in*: Proc. ZUM-95, Bowen, Hinchey (eds.), LNCS **967**, 264-275, Springer.

Swartout W., Balzer R. (1982); On the Inevitable Intertwining of Specification and Implementation. Comm. ACM **25**, 438-440.

van Dalen D. (1997); Logic and Structure. (3rd ed.), Springer.

Waldinger R., Blaine L., spinoza D., Gilham L-M., Goldberg A., Green C., Jullig R., Liu J., McDonald J., Smith D. R., Srinavas Y. V., Wang T. C., Westfold S. (1998); Specware Language Manual. Kestrel Institute. `http://www.kestrel.edu`

Weihrauch (2000); An Introduction to Computable Analysis. Springer.

Wirth N. (1971); The Development of Programs by Stepwise Refinement. Comm. ACM **14**, 221-227.

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

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

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