Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 67 (2007) 301–329
Science of Computer Programming
www.elsevier.com/locate/scico

Engineering and theoretical underpinnings of retrenchment

R. Banach[a,*], M. Poppleton[b], C. Jeske[a], S. Stepney[c]

[a] *School of Computer Science, University of Manchester, Manchester, M13 9PL, UK*
[b] *School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK*
[c] *Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK*

Received 3 July 2006; received in revised form 17 April 2007; accepted 20 April 2007
Available online 7 May 2007

Abstract

Refinement is reviewed, 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 a 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 proof obligation, simple examples, its use in requirements engineering and model evolution, and simulation properties. The interaction of retrenchment with refinement notions of correctness is overviewed, as is a range of other technical issues. Two case study scenarios are presented. One is a simple digital redesign control theory problem, and the other is an overview of the application of retrenchment to the Mondex Purse development.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Refinement; Retrenchment; Simulation; Requirements engineering; Model evolution; Digital redesign; Mondex purse

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 formal 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 many researchers and held as selfevident by practitioners.[1] The main motivation for the subject of this paper, retrenchment, is to help to assuage this dissonance. Retrenchment, which was first introduced in [23], in the specific context of the B-Method [2,147,97,129], 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

* Corresponding author. Tel.: +44 1612755720; fax: +44 1612756204.
E-mail addresses: banach@cs.man.ac.uk (R. Banach), mrp@ecs.soton.ac.uk (M. Poppleton), cj@cs.man.ac.uk (C. Jeske), susan@cs.york.ac.uk (S. Stepney).

[1] Many private conversations with members of both communities, not to mention reactions of numerous anonymous referees of varying backgrounds, have convinced us of the truth of this.

0167-6423/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2007.04.002

by its help. The reasons *why* one might want to include such pairs of models within a single development can be varied; we touch on some of these later.

This paper presents the retrenchment concept from scratch in a simple transition system framework.[2] It starts by observing that, over the years, refinement has been used not only as a method of implementation which guarantees 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 [10,11,14,148,109,108,15], these questions of emphasis are not always at the fore.

Various refinement notions also appear in more applications oriented specification notations, such as Z or VDM [132,76,89,90,146]. Not all of these are compatible, in the sense that a development step which is a refinement according to one notion may not be one according to another; see [57,53] for some relevant discussion. But a development step that fails to be a refinement on such a 'legal technicality' may well be a perfectly reasonable one from a systems engineering point of view. Such situations give rise to the need for a richer formal development notion, one that is capable of capturing such steps and 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 a specification constructor from refinement as an 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 an example and pause to define various simulation theoretic notions. In Section 3 we continue the small example to illustrate some of the problems that can arise during development steps if we adhere strictly to the formal definition of refinement as the sole method of passing from more abstract to more concrete models. Aspects of size, and management concerns, also rear their head in this regard. Having presented our motivations, Section 4 defines retrenchment itself, justifying the key proof obligation, recasting the running example via retrenchment, introducing output retrenchment, discussing how retrenchment relates to other work in the literature that addresses the limitations imposed by the 'refinement straitjacket', and considering how retrenchment as defined, also addresses wider issues of requirements engineering and model evolution. Section 5 revisits simulation in the context of retrenchment, explaining how it has to be approached in a different manner to refinement simulation. Section 6 records some simple default relationships between retrenchment and refinement. In Section 7, we outline how retrenchment ought to interact with the large number of notions of correctness that are to be found in the refinement literature. Section 8 outlines a selection of other technical issues regarding retrenchment and indicates their relevance; of particular note is the Tower Pattern, a structure that resonates in a number of places in the paper. In Section 9 we consider case study scenarios as they apply to 'real engineering' situations. 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 continuous–discrete control theory problem in detail. We then overview the way that retrenchment can contribute towards giving a more complete treatment of the Mondex Purse. This was a critical development, undertaken using Z refinement, which resides entirely in the discrete domain, but which nevertheless dealt with a number of modelling issues in a less than ideal way due to the exigencies of the Z refinement. Section 10 concludes.

2. Refinement, and requirements

Back in the days of [145,59,81], life was simpler, in that it was clear what was intended 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 imperceptibly, life got more complicated. The process of refinement in the preceding sense typically involves the incorporation/adoption of lower level detail into the lower level model. Indeed, one of the

[2] These days, latest developments are indicated on the Retrenchment Homepage [122].

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 detriment of requirements considerations. Indeed even in many textbook-scale examples of refinement, the more abstract models are sometimes 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.[3]

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. Different uses of refinement

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. (Note that the A subscript is a meta level tag, suppressed when inappropriate.) The operations will work on a state space U, having typical element u. For an $Op_\mathrm{A} \in \mathsf{Ops}_\mathrm{A}$, there are also input and output spaces $\mathsf{I}_{Op_\mathrm{A}}$ and $\mathsf{O}_{Op_\mathrm{A}}$ with typical elements i, o respectively (the anticipated subscripts on i and o to indicate the relevant Op_A are routinely suppressed). Primes, indices, etc. will be used to distinguish different elements of the same space. Initial states are defined as those that satisfy the property $Init_\mathrm{A}(u')$. In this paper, we 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 $u\text{-}(i, Op_\mathrm{A}, o)\!\!\rightarrowtail u'$, where u and u' are the before and after states, and i and o are the input and output values. An execution fragment of the *Abs* system is a finite or infinite sequence of contiguous steps, written $[u_0\text{-}(i_0, Op_{\mathrm{A},0}, o_1)\!\!\rightarrowtail u_1\text{-}(i_1, Op_{\mathrm{A},1}, o_2)\!\!\rightarrowtail u_2 \ldots]$, and drawn from the collection of abstract step relations $\bigcup\{stp_{Op_\mathrm{A}} | Op_\mathrm{A} \in \mathsf{Ops}_\mathrm{A}\}$. An execution fragment such that $Init_\mathrm{A}(u_0)$ 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 $Op_\mathrm{C} \in \mathsf{Ops}_\mathrm{C}$. States are $v \in \mathsf{V}$, inputs $j \in \mathsf{J}$, outputs $p \in \mathsf{P}$. Initial states satisfy $Init_\mathrm{C}(v')$. Transitions are $v\text{-}(j, Op_\mathrm{C}, p)\!\!\rightarrowtail v'$, which are elements of the step relation $stp_{Op_\mathrm{C}}(v, j, v', p)$.

In keeping with the overwhelming majority of applications, in this paper we stay within a forward simulation formulation of refinement. (See [57,53] for surveys 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 and so Op_A is identified with Op_C.

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

$$Init_\mathrm{C}(v') \Rightarrow (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge G(u', v')) \tag{2.1}$$

[3] A number of refinement theorists have confirmed this to us.

and for every corresponding operation pair Op_{A} and Op_{C}, the abstract and concrete step relations must satisfy:

$$G(u, v) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \Rightarrow (\exists\, u', i, o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge G(u', v') \wedge In_{Op}(i, j) \wedge Out_{Op}(o, p)) \quad (2.2)$$

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:

$$mset \ \text{-}(n, put_{\mathrm{A}})\!\!\twoheadrightarrow mset{+}\{n\} \ ; \ mset{+}\{n\} \ \text{-}(get_{\mathrm{A}}, n)\!\!\twoheadrightarrow mset. \quad (2.3)$$

As usual, we can refine the multiset to a sequence *mseq*, with *put* and *get* becoming obvious list manipulation operations (where @ is 'append' and :: is 'cons'):

$$mseq \ \text{-}(n, put_{\mathrm{C}})\!\!\twoheadrightarrow mseq@[n] \ ; \ n{::}mseq \ \text{-}(get_{\mathrm{C}}, n)\!\!\twoheadrightarrow mseq. \quad (2.4)$$

The retrieve relation for these is:

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

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

$$Init_{\mathrm{A}}(\varnothing) \quad \text{and} \quad Init_{\mathrm{C}}([]) \quad (2.6)$$

provide a suitable initialisation, and that the POs (2.1) and (2.2) hold.

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 sequence of elements output via get_{C} is unfailingly a prefix of the sequence 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 did not 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 [104,96,128].

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 [12,72,92,13]. 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):

$$tickvar \ \text{-}(tick_{\mathrm{C}})\!\!\twoheadrightarrow tickvar{+}1 \quad (2.7)$$

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):

$$(bins, tickvar) \ \text{-}(n, put_{\mathrm{C}})\!\!\twoheadrightarrow (bins \lhd\!\!+ \{tickvar \mapsto (bins(tickvar){+}\{n\})\}, tickvar)$$

$$(\{0\ldots k{-}1\}\times\varnothing\cup\{k\mapsto abin{+}\{n\}\}\cup\{k{+}1\mapsto bins(k{+}1)\}\ldots,tickvar)$$
$$\text{-}(get_{\mathrm{C}},n)\text{-}\!\!\succ$$
$$(\{0\ldots k{-}1\}\times\varnothing\cup\{k\mapsto abin\}\cup\{k{+}1\mapsto bins(k{+}1)\}\ldots,tickvar). \tag{2.8}$$

The corresponding retrieve relation will be:

$$G(mset,(bins,tickvar))\equiv\left(\sum_{k\in\mathsf{NAT}}bins(k)=mset\right) \tag{2.9}$$

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 (e.g. the details of $tick_{\mathrm{C}}$, and the way the distribution of invocations of $tick_{\mathrm{C}}$ 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 skip 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. Black, grey, and glass boxes

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 [133,142], 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 e.g. [67,68,66,69]). In another context one can mention the EXT operator of LOTOS refinement as manifestly addressing the specification constructor task [141,44,45].

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 implementer. We will call this the contracted model.[4] 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 accommodation 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. Since the users' perspective is already captured within the contracted model, how this is turned by the implementer into a running system need not concern them. 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. (A near ideal incarnation of the black box view of refinement is embodied in the Perfect Developer tool [52], in which refinement is invoked automatically. Other tools which create implementations by using code generators can be understood as doing essentially the same thing.)

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?[5] 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.

[4] For the sake of being able to discuss a simple scenario, we assume that there *is* a contracted model in the development. Realistic developments may have a more involved structure. For example, the Event B methodology [123,3–5] can be seen as an advanced case of system construction by refinement, in which structure is added via a(n often long) sequence of refinements, typically of the superposition kind. The difference between Event B refinements and traditional superposition refinements is that the new events added during refinement are not regarded as hidden, but address genuine system requirements. In this case the contracted model has to be seen as the last of the models, the one from which the code is generated.

[5] 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. See also the discussion in Section 4.4.

2.3. Simulation properties

We will now look at some simulation theoretic aspects of refinement, primarily to provide a contrast with the corresponding situation for retrenchment in Section 5. Suppose we have a refinement given by operation names $\mathsf{Ops}_{\mathrm{A}} = \mathsf{Ops}_{\mathrm{C}}$, retrieve relation G, and In_{Op}, Out_{Op} relations for each $Op \in \mathsf{Ops}$.

Definition 2.1. With the usual notations, let $u\text{-}(i, Op_{\mathrm{A}}, o)\!\!\rightarrowtail u'$ be an abstract step and $v\text{-}(j, Op_{\mathrm{C}}, p)\!\!\rightarrowtail v'$ a concrete step. Then the abstract step simulates the concrete step iff we have:

$$G(u, v) \wedge In_{Op}(i, j) \wedge G(u', v') \wedge Out_{Op}(o, p). \tag{2.10}$$

Since the relation (2.10) is symmetrical between abstract and concrete, we can just as easily say that the concrete step simulates the abstract step. In general, the two steps are said to be in simulation.

Definition 2.2. Suppose that $\mathcal{S} = [u_0\text{-}(i_0, Op_{\mathrm{A},0}, o_1)\!\!\rightarrowtail u_1\text{-}(i_1, Op_{\mathrm{A},1}, o_2)\!\!\rightarrowtail u_2 \ldots]$ and $\mathcal{T} = [v_0\text{-}(j_0, Op_{\mathrm{C},0}, p_1)\!\!\rightarrowtail v_1\text{-}(j_1, Op_{\mathrm{C},1}, p_2)\!\!\rightarrowtail v_2 \ldots]$ 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. Then $\mathcal{S}$ is a stepwise simulation of $\mathcal{T}$ iff (2.10) holds for each corresponding pair of steps of $\mathcal{S}$ and $\mathcal{T}$.

Stepwise simulation plays a pivotal role in relating properties of two systems related by refinement (as we have introduced it). We review briefly here some other familiar facts regarding simulation in refinement.

Proposition 2.3. *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 $\mathcal{S} = [u_0\text{-}(i_0, Op_{\mathrm{A},0}, o_1)\!\!\rightarrowtail u_1\text{-}(i_1, Op_{\mathrm{A},1}, o_2)\!\!\rightarrowtail u_2 \ldots]$ 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:

$$\begin{aligned} &\Theta_{\mathrm{S}}(u, v) \equiv G(u, v) \\ &\Theta_{\mathrm{L}}((i, Op_{\mathrm{A}}, o), (j, Op_{\mathrm{C}}, p)) \equiv In_{Op}(i, j) \wedge Out_{Op}(o, p) \end{aligned} \tag{2.11}$$

Definition 2.4. In the context of (2.11), the Abs transition system strongly simulates the Conc transition system iff:

$$Init_{\mathrm{C}}(v') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge \Theta_{\mathrm{S}}(u', v')) \tag{2.12}$$

and for all reachable concrete states v:

$$\begin{aligned} &\Theta_{\mathrm{S}}(u, v) \wedge v\text{-}(j, Op_{\mathrm{C}}, p)\!\!\rightarrowtail v' \Rightarrow (\exists\, u', i, o \bullet u\text{-}(i, Op_{\mathrm{A}}, o)\!\!\rightarrowtail u' \wedge \\ &\quad \Theta_{\mathrm{L}}((i, Op_{\mathrm{A}}, o), (j, Op_{\mathrm{C}}, p)) \wedge \Theta_{\mathrm{S}}(u', v')). \end{aligned} \tag{2.13}$$

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

Proposition 2.5. *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 [140,9,83]. 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 (that every L_1-theorem in T_1 is already a theorem in T_2 is normally immediate). We call an extension semiconservative if the reverse implication need not necessarily hold. We make an analogy between derivations in logical systems and sequences of execution steps in a transition system, according to the correspondence:

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

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:

$$G(u_0, v_0) \wedge G(u_f, v_f) \wedge Ins(is, js) \wedge Outs(os, ps) \tag{2.15}$$

holds, where *Ins* and *Outs* are suitable relations on the input and output sequences as a whole. 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 2.6. *Let* Conc *refine* Abs*. Then* Conc *is a semiconservative extension of* Abs*.*

The proof is a simple corollary of Proposition 2.3.

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 e.g. [1,95]. The notion also forms the approach to refinement espoused in the ASM formalism. See e.g. [38,36,39,40,37,126,127,134,41]. In specific examples ad hoc techniques are often required.

3. Shortcomings of refinement as a specification 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 sometimes 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:

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

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:

$$mseq \ \text{-}(n, put_C)\twoheadrightarrow mseq, \qquad \text{where length}(mseq) = 10. \tag{3.2}$$

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

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

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

but then the concrete and abstract *get* operations would break the refinement in the $\text{length}(mseq) = 10$ 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) do not exist:

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

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:

$$\begin{aligned} &mseq \ \text{-}(n, put_{\mathrm{C}}, \mathrm{FULL})\!\!\rightarrow Oflow, \quad \text{where length}(mseq) = 10;\\ &mseq \ \text{-}(get_{\mathrm{C}}, \mathrm{EMPTY})\!\!\rightarrow Uflow, \quad \text{where length}(mseq) = 0. \end{aligned} \tag{3.5}$$

To recover the system once it lands in one of these states we introduce a $reset_{\mathrm{C}}$ operation with transitions:

$$Oflow \ \text{-}(reset_{\mathrm{C}}, \mathrm{OK})\!\!\rightarrow [], \qquad Uflow \ \text{-}(reset_{\mathrm{C}}, \mathrm{OK})\!\!\rightarrow []. \tag{3.6}$$

Since we intend the $reset_{\mathrm{C}}$ 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 a refinement of the abstract (2.3).

Given our theme of arriving at a contracted model and then refining it to implementation, since the above cannot be accommodated via refinement, it has to belong to the above contracted model phases of development. However, one can take another view. We could redo the above in two stages: the first being the refinement of multisets to finite sequences as noted, the second the imposition of the other alterations. The second phase now become a post-refinement model change towards implementation, a different paradigm to our running one. Since both routes arrive at the same final model, they ought somehow to be compatible. Looking ahead a little, this will eventually prove to be the case, via the Tower Pattern, to be introduced in Section 8.

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 concerns 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 re-assembly of complex systems from more primitive components (if not necessarily intuitively simpler ones) more feasible. Using refinement to build up a complex system from 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 development 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 of various kinds, 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 contracted model becomes more difficult to read and to validate, and the costs of producing both the contracted model and the implementation manually and separately may become unacceptably high. (Probably the ideal solution to such a predicament is to use a tool like the Perfect Developer to generate the implementation automatically at little cost from the contracted model, as indicated earlier.)

Of course there is no intrinsic harm in having a 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 — it is just that it is more profitable if the contracted model is visibly simpler. In the same vein, the reverse engineering referred to earlier is not an entirely negative activity — the mental gymnastics involved in doing it will *always* lead to a deeper understanding of the desired system — it is just that there may be more profitable ways of employing the mental effort expended if the abstract model thus generated is far adrift of the system requirements.

3.3. Management aspects

Aside from the purely technical points indicated above, we note that a further consequence of size is that large developments are not undertaken in a vacuum. They happen in a context of customers, managers, and financial and other stakeholders. Most of these people are not preoccupied with technicalities, and take a different view of the intricacies of refinement than do the technical experts. Within the scope of a realistic development, situations such as the following can arise, all of which impede the successful application of refinement in one way or another.

Firstly, the customer may not permit a change in the specification (in order to make it refinable to the lower level models). The specification document serves several different purposes, including certification and validation, as well as formal development. Often a change to the specification would be more costly than getting the formal specialists to work around some perceived refinement limitation.

Secondly, real engineering applications never start from the blank sheet perspective of textbook or research paper examples and methods. This can make a completely purist approach to refinement unrealistic in practice.

Thirdly, a technical snag in a refinement development can show up when there is insufficient time left to redo things properly within existing schedules, deadlines and budgets. In a pure research environment the job would simply be considered unfinished and publication would be delayed. In an engineering environment, the job must be completed despite the snag, and engineering compromises become necessary.

Fourthly, even when they are permitted, changes to specifications arising from failures of refinement can depend on the detailed design route chosen. If a single specification is targeted at multiple platforms, such changes can easily be incompatible (e.g. consider two concrete versions of the multiset example above — done in a total correctness framework — featuring different bounds on *mseq*, say 10 and 20, and the problem of defining a single abstraction properly refinable to both of them). In such cases there may be no single abstract model that caters adequately for all the platforms.

Fifthly, specifications are not only used to initiate refinements, but also serve as an important means of communication between the various parties in a development. The semiconservative extension structure forced by refinement, may not always organise the system's requirements in a way that makes sense to domain experts. If it does not, then the refinement will not communicate as effectively as it should.

The points highlighted just now, and in the previous section, underline the usefulness of having a wider gamut of formal techniques, capable of addressing concerns that are vital either higher up the development hierarchy, or bite right at the bottom. Evidently for small systems these phenomena do not arise in a serious way.

4. Retrenchment

Having set out our motivations at some length above, we now come to retrenchment itself. The challenge in designing retrenchment was to come up with a notion that at minimum was expressive enough to be able to describe the problematic scenarios indicated above, while at the same time allowing as much meaningful contact with refinement theories as was practicable. Noting that the simulation based POs of typical refinement theories provide a very natural way of understanding the relationship between two models, retrenchment was designed by asking what modifications to these POs would address the stated goals. The approach fixed on was to enrich the relationship between models from one described by POs involving a single nontrivial relation (the retrieve relation $G(u, v)$) to one described by POs involving additional relations: 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 appears as a constraint in the antecedent, while the latter weakens the consequent.

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 it might be embedded in.

Note that the within relation P involves the inputs as well as the states. This allows a change of input representation and mixing of state and input information on the pre-side of a transition. Likewise the concedes relation C involves the outputs and allows change of output representation and mixing of state and output information on the post-side of a transition. Moreover before-states and inputs may appear in C too (after the semicolon, which in this instance is just an alternative punctuation), allowing a richer set of possibilities to be expressed by the concedes relation.

4.1. Basic concepts of primitive 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 point out that any such additional operations are *not* regarded as hidden, as they were at the end of Section 2.1.

To stay close to 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:

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

Note that the latter only makes sense for those operation names Op, common to both systems, and the subscripts on P_{Op} and C_{Op} indicate that each common Op can have a different P and C.

The POs define retrenchment in our framework, which is in contrast with the situation for refinement where the corresponding statements are derived from more abstract semantic or correctness considerations. We advance the following points in support of this — ultimately heuristic — design for the retrenchment operation PO.

Let us re-examine 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 that (from a black box perspective) no user of the system should be able to notice that it is not the abstract system doing the work. The $\forall$*Conc-step*$\exists$*Abs-step* ... structure of (2.2) makes sure that all concrete steps toe the line.

Retrenchment does not have this duty. It recognises that the abstract and concrete systems are incompatible in the strict sense that refinement demands, and seeks to illuminate the structure of the concrete system from the perspective of the abstract one in a glass box manner.

Regarding the issues that such a PO ought to cover, firstly we ought to allow many–many relationships between those abstract and concrete steps that are to be related. Secondly, for ease of mechanisation, and for ease of comparison and integration with refinement, it is useful to restrict ourselves to a standard shape of statement. Thirdly, incompatible models may contain parts that we do not need to relate, so we conjoin the within relation P to the rest of the antecedent, delimiting the reach of the PO as required. Permitting the abstract and concrete before-states and the abstract and concrete inputs to occur allows fine tuning of the before-configurations we wish to speak about, above

and beyond those identified by the retrieve relation G alone. Fourthly, incompatible models may contain parts that we need to relate, despite the fact that they do not strictly speaking re-establish the retrieve relation G, so we weaken the consequent 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. The form of the operation PO (4.2) supports all of these things.

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

If we examine the $\forall \mathit{Abs\text{-}step} \exists \mathit{Conc\text{-}step}(G \vee C)$ form, we must consider four things. Firstly, this form resembles a refinement PO from concrete to abstract systems, especially so if the effect of P and C is to strengthen applicability criteria and weaken the after-state criteria in the passage from abstract to concrete. Unfortunately, some of the difficult modelling situations we have to consider, such as passing between continuous and discrete models, are no better addressed by the opposite of refinement than by refinement itself, so no advantage is gained by this departure from the refinement-like form. Secondly, the $\forall \mathit{Abs\text{-}step}$ 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 (see Section 9.2 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 do not have a negative criterion that we must ensure the abstract system fulfils, as was the case for concrete systems in refinement. All of these considerations mitigate against adopting the $\forall \mathit{Abs\text{-}step} \exists \mathit{Conc\text{-}step}(G \vee C)$ form.

So we turn to the $\forall \mathit{Conc\text{-}step} \exists \mathit{Abs\text{-}step}(G \vee C)$ form. Here we consider three points. Point 1: we are not required to say something about all abstract steps, which in view of the remarks above is at least not detrimental. Point 2: we *are* required say something about all concrete steps, which helps to enhance the PO as a mechanism for the construction and description of the concrete system, which we thus regard as beneficial. In particular we must consider for any concrete step whether it should be: (a) excluded from consideration, because P is not valid there (or, to put it another way, that we ought to ensure that P is constructed in such a way that for such steps this is indeed the case); (b), included, but requires essential use of C to satisfy the PO; (c), included, but does not require C. Point 3 follows on from (c). In realistic practical cases, there may well be substantial subsets 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 (see comments following (4.8) below). When this arises, we are justified in viewing retrenchment as being 'like refinement except round the edges', i.e. like refinement for the majority of the state and I/O spaces, but not quite everywhere. In this regard, the fact that the retrenchment PO reduces to the refinement PO (and not to something else, as it would with the preceding form) is something we see as a real plus. 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 in Section 3.1, wherein refinement foundered, and show how it fits conveniently enough into the retrenchment framework. We recall:

$$
\begin{aligned}
&\{put_{\mathrm{A}}, get_{\mathrm{A}}\} = \mathsf{Ops}_{\mathrm{A}} \subseteq \mathsf{Ops}_{\mathrm{C}} = \{put_{\mathrm{C}}, get_{\mathrm{C}}, reset_{\mathrm{C}}\}\\
&\mathsf{U} = \mathcal{M}(\mathsf{NAT}),\ \mathsf{I}_{put_{\mathrm{A}}} = \mathsf{NAT},\ \mathsf{O}_{put_{\mathrm{A}}} = \varnothing,\ \mathsf{I}_{get_{\mathrm{A}}} = \varnothing,\ \mathsf{O}_{get_{\mathrm{A}}} = \mathsf{NAT},\\
&\mathsf{V} = \{ll \in seq(\mathsf{NAT}) | \mathrm{length}(ll) \leq 10\} \cup \{\mathit{Uflow}, \mathit{Oflow}\},\\
&\mathsf{J}_{put_{\mathrm{C}}} = \mathsf{NAT},\ \mathsf{P}_{put_{\mathrm{C}}} = \{\mathrm{FULL}\},\ \mathsf{J}_{get_{\mathrm{C}}} = \varnothing,\ \mathsf{P}_{get_{\mathrm{C}}} = \mathsf{NAT} \cup \{\mathrm{EMPTY}\},\\
&\mathsf{J}_{reset_{\mathrm{C}}} = \varnothing,\ \mathsf{P}_{reset_{\mathrm{C}}} = \{\mathrm{OK}\}.
\end{aligned}
\tag{4.3}
$$

Reprising the transitions:

$$
\begin{aligned}
&mset \ \text{-}(n, put_{\mathrm{A}})\text{-}\!\!\succ mset + \{n\}\\
&mset + \{n\} \ \text{-}(get_{\mathrm{A}}, n)\text{-}\!\!\succ mset
\end{aligned}
\tag{4.4}
$$

and

$$\begin{array}{ll} mseq \ \text{-}(n, put_{\mathrm{C}})\text{-}\!\!\succ mseq@[n] & \text{if length}(mseq) \leq 9 \\ mseq \ \text{-}(n, put_{\mathrm{C}}, \mathrm{FULL})\text{-}\!\!\succ Oflow & \text{if length}(mseq) = 10 \\ n :: mseq \ \text{-}(get_{\mathrm{C}}, n)\text{-}\!\!\succ mseq & \text{if length}(mseq) \leq 9 \\ mseq \ \text{-}(get_{\mathrm{C}}, \mathrm{EMPTY})\text{-}\!\!\succ Uflow & \text{if length}(mseq) = 0 \\ Oflow \ \text{-}(reset_{\mathrm{C}}, \mathrm{OK})\text{-}\!\!\succ [] & \\ Uflow \ \text{-}(reset_{\mathrm{C}}, \mathrm{OK})\text{-}\!\!\succ []. & \end{array} \tag{4.5}$$

The retrieve relation will be the original and transparent:

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

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

$$Init_{\mathrm{A}}(\varnothing) \quad \text{and} \quad Init_{\mathrm{C}}([]). \tag{4.7}$$

It remains to give the within and concedes relations for put_{C}, get_{C}. These are:

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

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 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 $\mathrm{length}(mseq) \leq 9$ then we could have reduced C_{put} to false: we would have excluded from consideration the part of put_{C} that behaves badly, at the cost of having diminished the scope of the relationship between the models that we had described. Since the concession trivialises in such a 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. In fact it would have reduced to an instance of conditional refinement (described for example in [49]) for put_{C}. 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 $\mathrm{length}(mseq) = 10$, both concrete and abstract *put* operations have something to do; the concedes relation describes the incompatibilities that ensue. However, when $\mathrm{length}(mseq) = 0$, the abstract *get* operation has no step; all that the retrenchment PO can do is to exclude those situations from consideration. One cannot expect the retrenchment PO to speak about parts of either system that have no counterpart in the other one.

The previous point emphasises that what does not fall within the scope of the retrenchment PO can be just as important as what does, and requires equal vigilance when retrenchment is used. In other words, a retrenchment must be thoroughly validated in the context of the relevant application domain. The discussion of glass boxes in Section 2.2, and the (a)–(c) taxonomy hinted at at the end of Section 4.1, all allude to the same thing. In particular, scrutiny of the domains and ranges of the various relations that make up a particular retrenchment constitutes the bare minimum that such a validation activity should consider.

4.3. Output retrenchment

In the preceding example, how are the outputs of get_{A} and get_{C} related for steps that re-establish the retrieve relation? They are equal, but nothing in (4.8) gives any hint about this. The shortcoming is easy enough to fix. We can replace the concedes relation C_{get} of (4.8) by:

$$C_{get}(mset', mseq', o, p; i, j, mset, mseq) \equiv (o = p). \tag{4.9}$$

We see that the inclusiveness of the disjunction in (4.2) allows both disjuncts to be true, and thus allows C to cover the salient facts about the outputs in the cases when both steps complete normally. In general, if all that is needed is some container for holding relevant facts about the development step being performed, then C will act as a suitable one, and the earlier primitive form of retrenchment is sufficient. However, it is often useful to separate the normally completing cases from the others. This arises when considering theoretical matters, and also when considering tool support for retrenchment. For this purpose we introduce the output relation $O_{Op}(o, p; u', v', i, j, u, v)$, which will be conjoined to $G(u', v')$ in the PO, generating the output form of retrenchment. As in C, the semicolon merely separates the most salient variables from others that are permitted, which in the case of O are o and p only since $G(u', v')$ is there to speak for the after-states (though u', v' can also occur in O if complex joint properties of outputs and states need to be expressed). The POs become:

$$Init_{\mathrm{C}}(v') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge G(u', v')) \tag{4.10}$$

$$\begin{aligned} &G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge ((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee \\ &\qquad C_{Op}(u', v', o, p; i, j, u, v))). \end{aligned} \tag{4.11}$$

Now instead of putting $(o = p)$ in the C_{get} as in (4.9), we can put it in O_{get} to arrive at a cleaner account of the development step in the example.

Note that there is no essential difference in expressivity between (4.2) and (4.11) if one configures the primitive retrenchment concession in the form $C_{\mathrm{prim}} \equiv ((G' \Rightarrow O_{Op}) \wedge (\neg G' \Rightarrow \text{etc.}))$. However, (4.11) is much more convenient when the separation of cases is desirable. Note further that the discussion following (4.2) justifying the shape of the PO applies unchanged for output retrenchment.

4.4. 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 [138], while our distinguishing the world above the contracted model from the one below is related to the 'open-closed principle' of [105].

The use of additional predicates during the refinement process appears in the assumption/commitment approach to formal development; see e.g. [56]. The rely/guarantee method of [88] is a specific model of this. It 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 [51,35], 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 [114], 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 [116,117] proposed program development using partial functions and a particularly convenient logic, prompted by finiteness and definedness considerations. Related to this approach is a body of work on partiality in algebraic specification, e.g. [46,47,91,48,74].

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 [77]. 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, [42,135,107,53] also discuss I/O refinement, designed to incorporate changes of I/O representation, while [43] discusses grey box refinement, voicing concerns related to ours.

The notion of observability itself has received more attention within the context of refinement in recent years, partly as a reaction to some of the issues raised here, noted also by others, partly in the search for greater flexibility. This allows more general refinements than we have (for expository convenience) admitted in this paper. The results of such refinements are processed through the observation machinery, and this determines the extent to which the phenomena playing a part in the refinement (states, I/O, or transitions) are to be regarded as corresponding to one another. See

[99,143,54,50] for a small selection of relevant work. Such developments can admit more intimate contacts between refinements and retrenchments, which, however, lie beyond the scope of this paper.

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 [100], 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 concedes relations strong enough. However, 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. And besides, in retrenchment there is a PO to discharge, which offers some scrutiny of the process.

By contrast, in [130,131], building on the work of [101], the concept of realisation approaches a similar challenge by effectively permitting the substitution of variables in a specification by fresh ones satisfying new properties. Compared to retrenchment, realisation wins a little and loses a little. It wins, since one can discover the fate of any property Π satisfied by the original specification, simply by applying the substitution to Π. It also loses, however, since the substitution effectively conjoins fresh properties, losing the disjunctive character of the retrenchment PO. Moreover, being a syntactic mechanism, it 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. Retrenchment, being rooted in the semantic arena, does not have the limitation indicated — see Section 9.2 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.

4.5. Retrenchment, functional requirements, model evolution

Thus far, retrenchment has been presented as a technique for addressing the problems of 'almost-refinement', which was its original aim. As with any theoretical structure though, one can forget its origins, and just dispassionately ask what purposes it might lend itself to. That there might be interesting answers to this, other than the original one, is hinted at by the fact that retrenchment is couched in logical terms via its principal PO, whereas the phrase 'almost-refinement' has overtones of nearness and farness that lie beyond the reach of a first-order statement like (4.2) or (4.11). So if retrenchment can address 'near-refinement' it should also be able to address 'nowhere-near-refinement'.

If a system *Abs* and a system *Conc* have operations that we identify (by matching their names for example), and there is a relation between their state spaces that we can identify as a retrieve relation, then we can use the mechanics of the retrenchment PO to express a provable relationship between the two models. This follows because, in extremis, we can make the within relation false — called the trivial retrenchment — obviating the need to prove anything at all. See Section 6. Of course, useful retrenchments arise from nontrivial within and other relations.

Such observations take retrenchment squarely into the fold of functional requirements engineering and model evolution. In this arena, retrenchment can express properties involving steps of the two models, which, though fairly general, still need to be provable via the PO. Once such properties have been identified, they then automatically benefit from the properties of retrenchments in general. Thus retrenchment brings to the activities of requirements engineering and model evolution a degree of mathematical rigour that can prove to be highly desirable. See for example [28,29]. Moreover, there is no hard boundary between the 'near-refinement' and 'nowhere-near-refinement' cases — such a situation as the movement from continuous modelling to discrete modelling is firmly *intended* as near-refinement, but its *mathematical embodiment* is rather closer to nowhere-near-refinement.

Regardless of whether it is deployed to address near-refinements, or nowhere-near-refinements, we reiterate that retrenchment must be performed with the active involvement of the human developers, i.e. it must be properly validated in the application context. Otherwise, the potential for unrestrained model change that it permits is just as prone to take a development in unprofitable as in profitable directions. Finally, it is worth noting that the human judgement aspect plays a powerful role in refinement also.[6] It is just that the requisite understanding has, with experience, become subsumed under 'good practice', enhancing the black box aspects of refinement noted above.

[6] Consider a concrete system built out of two copies of an abstract system, and duplicating all steps of the abstract system in both copies, in lockstep. Technically, this gives a nontrivial and unimpeachable refinement, but no one would regard it as a useful one.

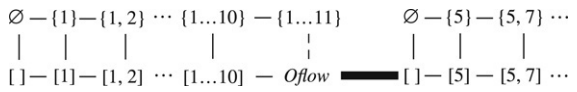

Fig. 1. A puncturated simulation scenario.

5. Simulation and retrenchment

In the context of retrenchment, simulation turns out to be a significantly different phenomenon compared with the refinement situation which was illustrated in Section 2.3. First we say what simulation for retrenchment is.

Definition 5.1. With the usual notations, let u -$(i,\ Op_{\mathrm{A}},\ o)\!\!\rightarrowtail u'$ be an abstract step and v -$(j,\ Op_{\mathrm{C}},\ p)\!\!\rightarrowtail v'$ a concrete step. Then the abstract step simulates the concrete step iff we have (for primitive retrenchment):

$$G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \wedge stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge (G(u', v') \vee C_{Op}(u', v', o, p; i, j, u, v)) \tag{5.1}$$

or for output retrenchment:

$$G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \wedge stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge ((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee C_{Op}(u', v', o, p; i, j, u, v)). \tag{5.2}$$

As earlier for refinement, the relations (5.1) and (5.2) (for primitive and output retrenchment respectively) are symmetric, and the two steps are said to be in simulation.

Since all the relations involved in the POs (4.2) or (4.11) are in principle partial, and the consequents of these POs contain the concedes relation disjunctively while the antecedents contain the within relation, the prospect for inductively deriving a stepwise simulation result like Proposition 2.3, that establishes (5.1) or (5.2) for each corresponding pair of steps, is gravely wounded. This blocks a standard route towards the semiconservative extension property, and hence a mechanism for relating system properties at the two levels.

One can overcome this to an extent by imposing additional restrictions on the two systems in question. In [26] there are some examples of this approach worked out in a formulation of retrenchment for the B-Method. Although such an approach can succeed technically, the generic character of the restrictions that are imposed tends to limit the usefulness of the results obtained, since they tend to be insensitive to the 'like refinement except round the edges' character of many practically relevant retrenchment scenarios. Thus the breakdown of the standard proof strategy for sequential compositions of retrenchment steps prompts a reappraisal of the role of simulation in retrenchment.

In retrenchment, the view of simulation as a symmetrical relation between *Abs* and *Conc* steps is more appropriate than the asymmetrical one arising from the retrenchment POs (4.2) and (4.11). Since retrenchment is primarily aimed at making quantitative the ways in which *Abs* and *Conc* *fail* to correspond cleanly to one another, it is of equal interest to see how properties of *Abs* do or do not translate to *Conc*, as it is to see how properties of *Conc* do or do not translate to *Abs*. Thus given a pair of steps s in *Abs* and t in *Conc* which are in simulation, then s may or may not have a step s' that can follow it[7] which is also simulable, and if there is such an s' and it is simulated by t' say, there is no guarantee that any such t' can be concatenated with t to form an execution fragment. Similar remarks apply if one starts the argument with t instead of s. And both arguments can be repeated in the reverse direction, considering predecessor steps of s and t. This variety of possibilities gives rise to a large number of unconventional simulation scenarios.

Example 5.2. Consider the example of Section 4.2, which admits the two scenarios illustrated in Figs. 1 and 2. In Fig. 1 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 $reset_{\mathrm{C}}$ operation in order to reinitialise; this is the thick transition in Fig. 1. 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

[7] That is, $[s, s']$ is an execution fragment.

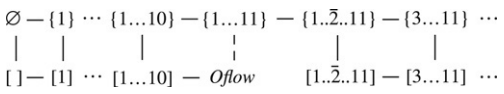

Fig. 2. Another puncturated simulation scenario.

again simulable and a simulation can be constructed as shown. Note that there is no connection between the abstract system's $\{1 \ldots 11\}$ state and the $\varnothing$ state which initiates the next portion of the 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 $\{1 \ldots 11\}$ and $\varnothing$ is stuttering [1].

In Fig. 2, after starting in like fashion with the abstract system again reaching $\{1 \ldots 11\}$, the abstract system this time get_{A}s the element 2 from the multiset, leaving $\{1..\bar{2}..11\}$. One concrete sequence that corresponds to $\{1..\bar{2}..11\}$, is $[1..\bar{2}..11]$, but neither this, nor any other concrete sequence corresponding to $\{1..\bar{2}..11\}$ is connected in any way with any of the preceding concrete sequences, nor indeed with the *Oflow* state, so this time there is a simulation hole that lies within the concrete system.

The cases in Example 5.2 show that, while conventional inductive reasoning based on an invariant cannot hope to control such situations, reasoning strategies more consciously targeted at termination, and based on the decrease of a suitable well founded variant, stand more chance on a case by case basis. This is an indication of the kind of ad hoc reasoning mentioned above.

Generalised simulation scenarios of the kind just illustrated are called punctured simulations; some early results on punctured simulation appear in [24] in the context of the B-Method. It turns out that the apparently rather anarchic punctured simulation landscape enjoys a surprisingly rich algebraic theory. See [18] for details.

Finally, it is fair to say that the simulation relation between two systems in a retrenchment captures the essence of what is being expressed by the retrenchment data (i.e. the within, concedes and other relations). In [20,19], fault trees are mechanistically extracted from the simulation relation of a suitable retrenchment, designed to capture the difference between an 'ideal' and a 'faulty' version of a system. It is conjectured that other analyses can be applied to the simulation relations of appropriate types of retrenchment, to give retrenchment based accounts of many more system engineering situations, especially of the nowhere-near-refinement kind.

6. Retrenchment and refinement default relationships

In this section we record some rather simple results on how retrenchment and refinement touch on one another. These results are in a sense like fenceposts, at the periphery of a territory in the interior of which interesting and practically useful retrenchments and refinements reside.

Proposition 6.1. *With the usual notations, let G and $\{In_{Op}, Out_{Op} | Op \in \mathsf{Ops}\}$, define a refinement from Abs to Conc. Then for any $\{P_{Op}, O_{Op}, C_{Op} | Op \in \mathsf{Ops}\}$ such that for all $Op \in \mathsf{Ops}$, $P_{Op} \Rightarrow In_{Op}$ and $Out_{Op} \Rightarrow O_{Op}$, G and $\{P_{Op}, O_{Op}, C_{Op} | Op \in \mathsf{Ops}\}$ define a retrenchment from Abs to Conc, called a retrenchment-extension of the refinement.*

Proposition 6.2. *With the usual notations, let G and $\{P_{Op}, O_{Op}, C_{Op} | O_p \in \mathsf{Ops}\}$ define a retrenchment from Abs to Conc, such that for all $O_p \in \mathsf{Ops}$, $P_{Op} \equiv (i = j)$, $O_{Op} \equiv (o = p)$, and $C_{Op} \equiv \mathsf{false}$. Then the retrenchment reduces to a refinement.*

Definition 6.3. With the usual notations, let G and $\{P_{Op}, O_{Op}, C_{Op} | O_p \in \mathsf{Ops}\}$ define a retrenchment from *Abs* to *Conc*, such that for all $O_p \in \mathsf{Ops}$, $P_{Op} \equiv \mathsf{false}$ and/or $C_{Op} \equiv \mathsf{true}$. Then the retrenchment is called a trivial retrenchment.

Proposition 6.4. *With the usual notations, let Abs and Conc be two systems with $\mathsf{Ops}_{\mathrm{A}} \subseteq \mathsf{Ops}_{\mathrm{C}}$, let G be a retrieve relation between their state spaces, and let $\{P_{Op}(i, j, u, v), O_{Op}(o, p; u', v', i, j, u, v) | Op \in \mathsf{Ops}_{\mathrm{A}}\}$ be arbitrary relations in the variables stated. Let $\{P^{\mathrm{Def}}_{Op}, C^{\mathrm{Def}}_{Op} | Op \in \mathsf{Ops}_{\mathrm{A}}\}$ be given by:*

$$P^{\mathrm{Def}}_{Op}(i, j, u, v) \equiv (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge (\exists\, u', o, v', p \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p))) \tag{6.1}$$

$$C^{\mathrm{Def}}_{Op}(u', v', o, p; i, j, u, v) \equiv (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \wedge \neg(G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v))). \tag{6.2}$$

Then G and $\{P^{\mathrm{Def}}_{Op}, O_{Op}, C^{\mathrm{Def}}_{Op} | Op \in \mathsf{Ops}_{\mathrm{A}}\}$ define a retrenchment from Abs *to* Conc*, called the default retrenchment from* Abs *to* Conc*.*

The default retrenchment embodies the observation that 'retrenchment is (more or less) able to relate any pair of systems'. While true as far as it goes, such a statement tends to disregard the fact — emphasised above — that retrenchment is useful only when undertaken in a glass box manner. In this sense, retrenchment is analogous to natural language. The fact that one can in principle babble incoherently does not detract from the usefulness of purposeful speech.

7. Notions of correctness

Up to now, for clarity, we have remained in a partial correctness framework, which is to say that the notion of retrenchment we have developed has been an extrapolation from a notion of refinement in which the relationship between abstract and concrete operations has been characterised by the implication (2.2) and nothing more. Unfortunately this approach admits a concrete model without any transitions at all, which is unsatisfactory for a black box version of refinement. To address this, various notions of total and general correctness have been developed in the refinement literature over the years.

For instance [57] covers the most commonly encountered positions on the partial and total correctness frameworks from a contemporary perspective. Earlier, [115] gave an accessible survey of a variety of approaches before detailing his own version of events. We also mention [60,78,2,82,15] which describe program development methodologies of this genre, and we recall that the more industrially targeted Z/VDM/RAISE techniques [132,146,53,89,70,113] 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 [61,137,55,65,63,110,111,106,64].

Given the proliferation of subtly different approaches, the question arises as to how retrenchment ought to relate to them. For instance, ought there to be a separate version of retrenchment for each individual formulation of refinement, or not? A number of clues to help answer this question suggest themselves.

Firstly, and most importantly, in the context of refinement, correctness (of a more concrete model) is exactly the opposite of abstraction, and any specific notion of refinement amounts to a precise definition of these two concepts. That there is more than one such definition to choose from, simply attests to the fact that what is meant by for example abstraction in the informal human domain (even when focused on the kind of system we deal with) can be rather hazy, and does not necessarily correspond to any one of the precise definitions. So, since retrenchment is concerned with issues that require a departure from refinement, there is no a priori need for retrenchment to reflect correctness notions in a way that is obviously recognisable from refinement, i.e. *correctness (in the above technical sense) is not an issue per se for retrenchment*. Pursuing this line of argument suggests that it is only when refinement and retrenchment need to interact that the notion of correctness in force for refinement should impact the retrenchment.

Secondly, one way or another, the various approaches to correctness in refinement have in common an inclusion of a set of 'good' concrete transitions into appropriate abstract transitions, the inclusion being mediated by a suitable retrieve relation or its analogue. There are few if any disagreements about the properties of these good transitions among the various approaches, and this inclusion has as its retrenchment counterpart the operation PO (4.2) or (4.11). The differences between the various approaches concern the less good transitions. These are typically transitions that do not, in some sense, initiate or terminate properly, or that are not adequately enough related to the good transitions in the other model of the refinement. Given its application oriented remit, retrenchment ought to be focused predominantly on the impeccably behaved transitions, since it is those that will make up real systems. This insight offers an Occam's Razor for cutting through the proliferation of possibilities.

Thirdly, aside from the inclusion between the good transitions, notions of correctness usually involve sets or predicates in the before-values associated with operations: these typically have names such as *applicability, termination, feasibility*, etc., and these figure in the POs of the specific theory.[8] The good transitions are those which

[8] It is worth noting that, for many theories, the detailed form of the POs inhibits straightforward simulation results such as Proposition 2.3

satisfy all the relevant criteria pertaining to these sets or predicates, in contrast to any remaining possibilities admitted by the theory. In being focused on the good transitions, the transitions that a retrenchment speaks about ought thus to satisfy all the relevant criteria too.

On the above basis, we can formulate a widely applicable general policy for how retrenchment interacts with a broad class of correctness notions.

Suppose that we have a model of computation endowed with a notion of correct refinement that can be expressed using a number of predicates $\theta_1, \theta_2 \ldots \theta_n$ (aside from the inclusion of the good transitions). For a given system we assume that these will be predicates in the before-values. When a retrenchment between systems *Abs* and *Conc* needs to interact with such refinements, we stipulate that for each Op:

$$G(u, v) \wedge P_{Op}(i, j, u, v) \Rightarrow \theta_{\mathrm{A},1}(u, i) \wedge \cdots \wedge \theta_{\mathrm{A},n}(u, i) \wedge \theta_{\mathrm{C},1}(v, j) \wedge \cdots \wedge \theta_{\mathrm{C},n}(v, j) \tag{7.1}$$

must hold, where $\theta_{\mathrm{A},1} \ldots \theta_{\mathrm{A},n}$ and $\theta_{\mathrm{C},1} \ldots \theta_{\mathrm{C},n}$ are the predicates for the abstract and concrete system respectively. This says that those concrete transitions which satisfy the antecedent of the retrenchment PO will automatically be in the domain of 'good' transitions (and analogously for the relevant abstract transitions). Experience has shown that the principle (7.1) leads to clean interactions between retrenchment and refinement, in particular in the case of the Z notion of correct refinement [86]. Note however that (7.1) differs from (and is intended to supersede) the earlier approach to correctness in [23].

A side effect of the principle (7.1) is that it encourages (even if it cannot guarantee) that the notion of simulation arising from a retrenchment (i.e. (5.1) or (5.2)), coincides with the corresponding notion arising from the relevant notion of refinement. Thus, if some particular instance of (5.1) or (5.2) for a pair of abstract/concrete steps requires the validity of the concession in order to make (5.1) or (5.2) valid, then it is likely that such a simulation instance does not coincide with a refinement simulation instance. However, it may be possible that such a retrenchment simulation instance can be preceded by one or more retrenchment simulation instances in which the retrieve relation is re-established, to form a simulation between two fragments. For these preceding instances, the truth of (7.1) makes it more likely that they also satisfy the criteria for being refinement simulation instances.

8. Other technical issues for retrenchment

Above we have discussed a few of the technical issues that arise with retrenchment, and without going into a great deal of depth. In this section, we briefly indicate a selection of other important issues that will be dealt with elsewhere. The list is by no means exhaustive.

Composition mechanisms. Viewed purely as data structures, retrenchment is a richer one than refinement since it contains G, P_{Op}, O_{Op}, C_{Op}, whereas refinement has only G, (supplemented by suitable In_{Op}, Out_{Op} relations where necessary). Thus one can anticipate a wide variety of composition mechanisms for retrenchment, many of which trivialise when restricted to the refinement case. The presence of the disjunction in the consequent of the retrenchment PO necessitates care in defining composition mechanisms in order to assure associativity. These issues are explored in [22].

Stronger compositions. The presence of the disjunction in the retrenchment PO, together with the distributive law used in extracting composition mechanisms, can easily lead to a rapid proliferation of unproductive cases in a composed C. Judicious strengthening of the output and concedes relations by information from the PO antecedent can control these effects; however, associativity is made more difficult thereby. For vertical composition, these matters have been studied in some depth in [21,29].

General retrenchments. In this paper we examined primitive and output retrenchments, but there is no reason to not consider other propositional shapes for the PO consequent, nor indeed for its antecedent. General retrenchment is the study of these more flexible retrenchment techniques, potentially more suited to specific application areas where specific kinds of design information deserve to be singled out in the propositional structure of the adopted PO. Some easy cases have been studied, e.g. sharp retrenchment, with consequent shape $((G \vee C) \wedge V)$, [25]. The facts for sharp output retrenchment, with consequent shape $(((G \wedge O) \vee C) \wedge V)$, can easily be inferred. Associativity of composition for the POs in the general case is the major technical challenge.

Retrenchment and refinement interworking, and the Tower Pattern. In order to gain the benefit of the strengths of both techniques (reasoning control in refinement, expressive model evolution in retrenchment), a theory of their

interworking is needed. This considers firstly compositions of retrenchments and refinements, and allows various system design problems to be formulated as algebraic ones: for example how to translate a previously constructed refinement development through an evolution of one of its models described via a retrenchment. Such problems have been studied in [17,87], and most recently and comprehensively in [86]. The technical details can get surprisingly arduous, but they ultimately lead to the *Tower Pattern*, a commuting rectangular grid of (horizontal) retrenchments and (vertical) refinements, in which any path between two points represents the same retrenchment. This is a very widely applicable scheme for using retrenchment to bridge between refinement developments of an evolving system definition, or between refinement developments of the same system definition that take differing levels of real world detail into account. The fact that the grid commutes yields compatible views of the same evolution step as being either a high level design change compatibly propagated down through levels of refinement or a low level change for implementability, propagated up through levels of refinement to the highest level of abstraction. The mini development of Section 3.1 could be recast in this way; see Section 9.3 for a more extensive example.

Behavioural and temporal properties. In the face of the failure of the standard simulation theoretic techniques for addressing behavioural properties that are enjoyed by refinements, the raw simulation relation has to be studied in an ad hoc manner. Once this is available, a rich theory of system properties under retrenchment can be developed. See [18].

Coarse grained retrenchment. In all retrenchments thus far, one abstract step has been related to one concrete one. While this is a very useful perspective, it is not the only one, any more than single step simulation is the only perspective on refinement. An alternative perspective, in which several abstract steps can be related by retrenchment to several concrete ones, is liable to be even more interesting than in the refinement case, as the greater flexibility of retrenchment can encompass more varied coarse grained evolution of properties. The interaction between single step and multiple step retrenchments promises to be a very fruitful area. See [27].

9. Case study scenarios

In Section 4.2 we saw retrenchment applied to a simple example in the discrete domain. Such examples, though quite explicit, run the risk that, through their small size, they make only a feeble case for retrenchment, since with some easy to make adjustments a refinement can often be recovered, as noted in Section 3.2. In this section, therefore, we discuss the prospects for applying retrenchment in more demanding situations. Section 9.1 presents some background for cases involving traditional applied mathematics. Section 9.2 presents the essence of a simple control redesign example, showing that the retrenchment structure provides suitable containers for the mathematical facts that describe the situation. By contrast, Section 9.3 sketches how retrenchment can make a refinement based development of a purely discrete application (the Mondex Purse) more complete, by better taking into account issues that are awkward to handle properly with refinement.

9.1. Modelling the real world

Today, the proliferation of embedded digital controllers in physical systems requiring continuous mathematics for their description continues to grow. Many such systems have genuine safety connotations associated with their use (think of the millions of lines of code in a modern car). This growth increases the scope for underpinning such designs via formal techniques, in order to raise the level of dependability of the overall application. However, traditionally understood formal methods are often poorly suited to 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. This is not to say that one cannot build models of portions of continuous mathematics within logical and algebraic foundations — indeed the classical constructions of complexes via reals, rationals, integers, naturals à la Weierstrass and Cantor can be seen as underpinning this, and there are many contemporary research directions that can also be understood as assisting such an objective at least as a side effect, e.g. formal topology [124,125], computable reals [16], computable analysis [144], etc. However, the complexity of these undertakings denies their efficient adoption as a formal basis for applied mathematics in engineering applications, quite aside from their focus on foundational matters.

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. In fact, basic aspects of analysis have been examined from the mechanical theorem proving point of

view [75,71], and this work shows that a formal approach is entirely feasible, but is no small job. A different though comparable approach 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 every specific application.

Recently an intermediate position has become tenable, in which a core body of applicable continuous mathematics has been constructed, starting from a corrected version of the core PVS NASA libraries, verified from the ground up in the PVS theorem proving environment [98,112], PVS being a prover built with efficiency for applications uppermost in mind. The scale of the extant formal corpus (which required no less time to construct than indicated above) is such that one could conceive making a moderate further investment of effort to build, in a reasonable amount of time, sufficient specialised extensions to enable it to cover a development such as that described in Section 9.2. This opens the door to doing fully formal genuine engineering developments in the conventional applied mathematics sphere.

Such a two-stage approach is reminiscent of the way applied mathematics has been built up over the centuries. Rather than rederiving everything from first principles every time, 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. Done in a consistent way, a steadily increasing body of useful results could be accumulated, making subsequent development gradually easier.

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 some interest 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 discrete approximations can be depended on. Such results are prime candidates for inclusion in the formal justification of the appropriate development steps in real world applications where relevant. 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.

9.2. 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 in these so-called hybrid systems. See for example [8,103,58,93,139,79,102,7,6].

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 e.g. [94,73]. 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 this design space using refinement; even if it were possible, so much ingenuity would have to go into choosing the abstract model wisely in order that a refinement relation might arise, that the point of ever having the abstract model would largely disappear. More importantly, each change of discretisation scheme would entail the re-engineering 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 is justifiable on engineering grounds, is considered undesirable 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, which permits the *reuse* of the abstract model in different implementation scenarios.

We consider a control redesign situation, kept simple to prevent the scale of the enterprise from overwhelming the remainder of the paper.[9] The problem, posed in Laplace transform space as is common in engineering situations, is given by:

$$sx_{\mathrm{C}}(s) = A_{\mathrm{C}}x_{\mathrm{C}}(s) + B_{\mathrm{C}}r_{\mathrm{C}}(s) \tag{9.1}$$

where $x_{\mathrm{C}}(s)$ is the Laplace transform of the continuous system state, A_{C} and B_{C} are constants, and $r_{\mathrm{C}}(s)$ 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.[10]

Accordingly, in real time, the continuous system is described by the differential equation:

$$\dot{x}_{\mathrm{C}}(t) = A_{\mathrm{C}}x_{\mathrm{C}}(t) + B_{\mathrm{C}}r_{\mathrm{C}}(t) \tag{9.2}$$

where $\dot{x}_{\mathrm{C}}(t)$ is the time derivative of $x_{\mathrm{C}}(t)$.

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 formalised approaches to analysis mentioned in Section 9.1, we therefore see that unless they were 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 support most engineering applications.

One possible discrete system corresponding to (9.2) is given by:

$$\Delta^{+T}x_{\mathrm{D}}(k) = A_{\mathrm{D}}x_{\mathrm{D}}(k) + B_{\mathrm{D}}r_{\mathrm{D}}(k) \tag{9.3}$$

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

$$\Delta^{+T}x_{\mathrm{D}}(k) = \frac{x_{\mathrm{D}}(k+1) - x_{\mathrm{D}}(k)}{T} \tag{9.4}$$

and A_{D} and B_{D} are constants, r_{D} being the discrete external input signal. The solution of (9.3) for the next state is immediate:

$$x_{\mathrm{D}}(k+1) = (1 + TA_{\mathrm{D}})x_{\mathrm{D}}(k) + TB_{\mathrm{D}}r_{\mathrm{D}}(k) \tag{9.5}$$

while the solution of (9.2) is standard, and for a period T to the future of a starting point $t = kT$, is given by:

$$x_{\mathrm{c}}((k+1)T) = \mathrm{e}^{A_{\mathrm{C}}T}x_{\mathrm{C}}(kT) + \int_0^T \mathrm{e}^{A_c(T-\tau)}B_{\mathrm{C}}r_{\mathrm{C}}(kT+\tau)\mathrm{d}\tau. \tag{9.6}$$

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

[9] A similar but somewhat less primitive control redesign case study to this one is presented in [121].

[10] 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.

$$x_{\mathrm{C}}((k+1)T) = (1 + TA_{\mathrm{C}})x_{\mathrm{C}}(kT) + TB_{\mathrm{C}}r_{\mathrm{C}}(kT) + o(T) \tag{9.7}$$

so that:

$$\begin{aligned} x_{\mathrm{C}}((k+1)T) - x_{\mathrm{D}}(k+1) \;&=\; (x_{\mathrm{C}}(kT) - x_{\mathrm{D}}(k)) + T(A_{\mathrm{C}}x_{\mathrm{C}}(kT) - A_{\mathrm{D}}x_{\mathrm{D}}(k)) \\ &\quad + T(B_{\mathrm{C}}r_{\mathrm{C}}(kT) - B_{\mathrm{D}}r_{\mathrm{D}}(k)) + o(T) \end{aligned} \tag{9.8}$$

Now at time $(k+1)T$, the difference between the two states, $x_{\mathrm{C}}((k+1)T) - x_{\mathrm{D}}(k+1)$, becomes comparable to that at time kT, i.e. $x_{\mathrm{C}}(kT) - x_{\mathrm{D}}(k)$, provided:

$$A_{\mathrm{C}}x_{\mathrm{C}}(kT) - A_{\mathrm{D}}x_{\mathrm{D}}(k) + B_{\mathrm{C}}r_{\mathrm{C}}(kT) - B_{\mathrm{D}}r_{\mathrm{D}}(k) = o(1). \tag{9.9}$$

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 (9.5), being exact, remains unchanged, but (9.6) acquires more terms, which feed through to (9.8), and to (9.9), which becomes:

$$\begin{aligned} &A_{\mathrm{C}}x_{\mathrm{C}}(kT) - A_{\mathrm{D}}x_{\mathrm{D}}(k) + B_{\mathrm{C}}r_{\mathrm{C}}(kT) - B_{\mathrm{D}}r_{\mathrm{D}}(k) \\ &\quad + \frac{T}{2}(A_{\mathrm{C}}^2x_{\mathrm{C}}(kT) + A_{\mathrm{C}}B_{\mathrm{C}}r_{\mathrm{C}}(kT) + B_{\mathrm{C}}\dot{r}_{\mathrm{C}}(kT)) = o(T) \end{aligned} \tag{9.10}$$

where $\dot{r}_{\mathrm{C}}(kT)$ 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:

$$(x_{\mathrm{C}}(t), \dot{x}_{\mathrm{C}}(t)) \text{-}(r_{\mathrm{C}}(t), Op_{\mathrm{A}})\!\!\rightarrow (x_{\mathrm{C}}(t'), \dot{x}_{\mathrm{C}}(t')) \tag{9.11}$$

where t and t' are nonnegative real valued and $t < t'$; and there is a requirement that there exists a (global) solution of (9.2), such that for any $0 < t < t'$ the solution agrees with the quantities appearing in (9.11) at t and t'. Standard theory [85,80] says that there is enough information recorded in the state vector $(x_{\mathrm{C}}(t), \dot{x}_{\mathrm{C}}(t))$, to ensure that a unique analytic solution is determined by it and the input, on which (9.6) is based.

The concrete system is given by a transition relation for a concrete operation Op_{C} of the following kind:

$$x_{\mathrm{D}}(k) \text{-}(r_{\mathrm{D}}(k), Op_{\mathrm{C}})\!\!\rightarrow x_{\mathrm{D}}(k+1) \tag{9.12}$$

where k is natural valued and the quantities appearing in (9.12) must constitute a solution of (9.5). Clearly there is enough information recorded in the state $x_{\mathrm{D}}(k)$ to ensure that a unique solution to (9.5) is determined by it and the input.

As a simple retrieve relation, we will take:

$$G(x_{\mathrm{C}}(kT), x_{\mathrm{D}}(k)) \equiv |x_{\mathrm{C}}(kT) - x_{\mathrm{D}}(k)| \leq \varepsilon \tag{9.13}$$

where ε is sufficiently small (or indeed even zero). Now (9.9) lends itself to a retrenchment reinterpretation with within and concedes relations:

$$P_1(r_{\mathrm{C}}(kT), r_{\mathrm{D}}(k), x_{\mathrm{C}}(kT), x_{\mathrm{D}}(k)) \equiv |A_{\mathrm{C}}x_{\mathrm{C}}(kT) - A_{\mathrm{D}}x_{\mathrm{D}}(k) + B_{\mathrm{C}}r_{\mathrm{C}}(kT) - B_{\mathrm{D}}r_{\mathrm{D}}(k)| \leq o(1) \tag{9.14}$$

$$C_1(x_{\mathrm{C}}((k+1)T), x_{\mathrm{D}}(k+1); r_{\mathrm{C}}(kT), r_{\mathrm{D}}(k), x_{\mathrm{C}}(kT), x_{\mathrm{D}}(k)) \equiv |x_{\mathrm{C}}((k+1)T) - x_{\mathrm{D}}(k+1)| \leq o(1). \tag{9.15}$$

If we wish to extend the argument to second order we may do so, though we would have to include $\dot{r}_{\mathrm{C}}(kT)$ as an additional input in the formulation of (9.11) because of its presence in (9.10). Assuming this, (9.10) yields:

$$\begin{aligned} P_2(r_{\mathrm{C}}(kT), r_{\mathrm{D}}(k), x_{\mathrm{C}}(kT), x_{\mathrm{D}}(k)) \equiv \Big| &A_{\mathrm{C}}x_{\mathrm{C}}(kT) - A_{\mathrm{D}}x_{\mathrm{D}}(k) + B_{\mathrm{C}}r_{\mathrm{C}}(kT) - B_{\mathrm{D}}r_{\mathrm{D}}(k) \\ &+ \frac{T}{2}(A_{\mathrm{C}}^2x_{\mathrm{C}}(kT) + A_{\mathrm{C}}B_{\mathrm{C}}r_{\mathrm{C}}(kT) + B_{\mathrm{C}}\dot{r}_{\mathrm{C}}(kT))\Big| \leq o(T) \end{aligned} \tag{9.16}$$

$$C_2(x_C((k+1)T), x_D(k+1); r_C(kT), r_D(k), x_C(kT), x_D(k)) \equiv |x_C((k+1)T) - x_D(k+1)| \leq o(T). \tag{9.17}$$

As (primitive) 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 impacting the abstract model. The fact that this is possible is one of the major strengths of the retrenchment approach.

9.3. The Mondex purse and retrenchment

The Mondex Purse is a smartcard electronic purse for containing genuine money. As such, it is a security critical application, and the 1990s development of Mondex was one of the first developments to achieve the highest possible ITSEC rating of E6 (equivalent these days to a Common Criteria rating of EAL7) [62,84]. The ITSEC E6 rating requires there to be an abstract model, a concrete model, and a proof of correspondence between them; in the case of Mondex, this proof was a human-performed refinement proof between abstract and concrete models written in Z. The Mondex project generated a public version [136], of the models and proof in the more comprehensive commercially sensitive development. The development as described in [136] remains an impressive achievement, not least in being a trailblazer for showing that fully formal techniques could be applied within realistic time and cost limitations on industrial scale applications.

The Mondex development consists of two refinements, necessitating three models: the A(bstract) model, the B(etween) model, and the C(oncrete) model. The A model is a model of atomic funds transfer, which can either complete successfully (lodging the funds transferred instantaneously in the destination purse), or atomically 'lose' the funds (placing them in a special 'lost' component of the state). The B model captures the essence of the funds transfer protocol, and is thus non-atomic. The protocol can either succeed or fail, corresponding to the two kinds of A model outcome. The A model is refined to the B model, and the non-atomicity raises the issue of the resolution of nondeterminism within the refinement. In fact the abstract nondeterminism is resolved early with respect to the protocol, forcing the A to B refinement to be a backward one. The backwards direction of the refinement in turn necessitates the B model state to be constrained by a number of properties (mainly concerning the ether of messages in transit between purses), in order to maintain the integrity of the protocol. This means that the B model cannot be taken verbatim as a representation of the real world environment of the protocol, since the constraints do not correspond to unavoidable properties of the raw B model state. Fortunately, the constraints are inductive properties (over runs of the protocol) of the unconstrained state, so the C model is a version of the B model without the constraints, and a forward refinement from the B model to the C model establishes that all states reachable via the protocol in the unconstrained world nevertheless satisfy the B model constraints. The overall Mondex refinement is thus the composition of the A to B and B to C refinements. This state of affairs, the content of [136], is summarised in the left-hand column of Fig. 3.

Despite the impeccable credentials of the Mondex development as a genuine and honest representation of the development within a fully formal framework, the exigencies of refinement (to which we have often referred above) caused a number of issues to be treated in a less than ideal manner. In the Mondex project itself, the treatment of these issues was dealt with via suitable informal arguments to justify the stance taken, but it would clearly have been better to be able to integrate the treatments and their supporting arguments into the refinement development in a suitably formal manner.

Below, we summarise the issues themselves, and how retrenchment can achieve the desired integration. Instrumental to the integration exercise for most of the cases is the Tower Pattern (see Section 8). In Fig. 3 we

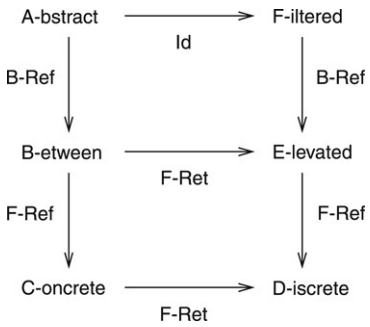

Fig. 3. The Tower Pattern instantiated for the Mondex development.

illustrate how the original refinement chain from A to C can be related to a further refinement chain from F to D via a collection of retrenchments, forming an instance of the Tower. The typical scenario runs as follows. An issue is identified as being imperfectly treated in the development. Since the top level is atomic, such issues are typically not visible there, and they emerge only in the lower level models. Accordingly, a more accurate low level model can be constructed via a retrenchment from the C model. This yields the D(iscrete) model, so named for the finiteness of the data structures that are to be found there. The B to C refinement and the C to D retrenchment can now be composed, yielding a retrenchment from B to D. Using results from [86], this retrenchment can now be factored the other way, into a retrenchment followed by a refinement, in a canonical way. This construction yields the E(levated) model, which lifts the level of abstraction of the C to D retrenchment to that of the B model. Due to the particularly careful design of the original refinement, in most cases, a refinement can now be discerned from the original A model to the E model, enabling us to complete the Tower by making the F model be a copy of the A model.

Sequence number. The integrity of the low level protocol depends partly on the sequence number of the transaction in progress. Sequence numbers are not required in the A model, but are vital in the B and C models, where they are naturals. In the actual Mondex implementation they are bounded numbers, although the bound is large. In this scenario, one can take into account the boundedness of the real sequence numbers via a straightforward retrenchment from the C model to the D model; the simple example of Section 4.2 indicates the general nature of what is needed (though the C to D retrenchment is even simpler since sequence numbers only ever increase). The retrenchment can now be lifted to the E model as indicated above (it turns out that the E model can be taken to be a close analogue of the D model), and the clean design of the original refinement, and the fact that sequence numbers are absent from the abstract model, enables the Tower to be completed rather trivially with the A and F models identical. See [30].

Log full. Transfers completing abnormally in the concrete models are aborted and logged locally by purses. The relationship between local purse logs and the 'All value accounted' security property (which relates successful and unsuccessful concrete transfers to the abstract balances and 'lost' components respectively) is rather complicated. Suffice it to say here that purses' log contents are vital. Logs occur in the B and C models where they are unbounded; in reality they are finite, and the bound is small. From a purely mathematical perspective, although there are minor differences in the way that the modelling is done in the D model, there are strong analogues between this scenario and the preceding one: both deal with finite capacity situations. Thus the construction of the E and F models follows the same pattern as above. See [31].

The fascinating aspect of these two scenarios concerns the validation of the respective retrenchments necessitated by the demands of a properly supported glass box design activity. In the sequence number case the bound was large, and validation focused on confirming that the actual bound would not ever be reached during any realistic use of the purse. In the full log case the bound was small, and validation focused on ensuring that no fresh transaction could start when the log was full, in case it too failed, creating a log entry for which there was no room. The retrenchment approach was able to identify out common (or nearly common) structural aspects of these two scenarios, while recognising (via the glass box philosophy) that their appropriateness could not be justified on the relatively simple structural forms that retrenchment offers, and would always depend on application specific criteria.

Hash function. The concrete B and C models implement the abstract 'lost' component in terms of an off-card central archive into which purses' log contents are saved. A purse needs to be assured that the data is safely in the archive before it can clear it from its own, highly constrained, log memory. Safe archival is signalled to the purse using a CLEAR code. The purse log contents are assumed to be in total injective correspondence with the CLEAR

codes, as that property is required in the proof of the maintenance of the security properties. In reality of course a cryptographic hash function is used, which is not injective, but was informally argued to be 'sufficiently injective' in the actual Mondex implementation. In this scenario one can use retrenchment to contrast the ideal situation of the C model injective function with the realistic hash function of the D model.

Consider the following state of affairs surrounding the log clear operation in the D model. Before log clearance, the (set theoretic) union of the central archive and local purse log is correct, and therefore the relevant security invariants hold. Suppose nevertheless that there is a discrepancy (masked by the union) between the central archive and the log, which has arisen due to some past activity, and such that the archive is incorrect. Suppose that a CLEAR code, generated from the archive, arrives at the purse, and suppose that it is found by the purse to correspond to its log contents, even though these do not match the archive, (an eventuality made possible by the many-one nature of the hash function). Then the log clear operation will make the security invariants fail in the D model, since the log/archive discrepancy will become unmasked during log clearance. Such a turn of events is impossible in the C model, so a C to D retrenchment of the log clear operation can capture the above scenario in the concession, while giving an account of the normal playout of log clearance in the output relation. The C to D retrenchment can then be lifted to the E model as before.

The final lifting of the E model to the F model is a little more interesting in this scenario. In the conventional Modex world, money is never truly lost, since the money contained in protocol messages that fail to arrive can be reliably recovered, due to the properties of the logging/archiving parts of the protocol. So 'lost' really means 'recoverably lost' in the A, B, C models. The scenario we described in the D model introduces an additional category of 'irrecoverably lost' funds, those inadvertently discarded from the log even though they are not in the archive. To deal with this at the level of the F to E retrieve relation, it is sufficient to replace an equality (between the F and E recoverably lost funds) with an inequality, and to thereby complete the tower construction. See [32].

Balance enquiry. Each purse has a balance enquiry operation; consider the following scenario for invoking it in the middle of a B (or C) model transaction. Recall that the resolution of nondeterminism in the A model is synchronised with the beginning of (the critical portion of) the B model protocol in the backward refinement. Therefore both models agree about when the transaction starts. They disagree however about when the transaction ends. In the A model it ends immediately, while in the B model it does not end until the message carrying the money arrives. Suppose we perform a balance enquiry for the recipient purse while the money is in flight. In the A model, the money is already there and incorporated in the A model balance output. In the B model it is yet to arrive. So the A and B balance outputs will disagree. In the actual Mondex implementation this is handled formally by a modelling trick, using finalisation instead of the enquiry operation output to observe the state. Taking into account also the other details of the backward refinement, the resulting treatment of balance enquiries can appear so very counterintuitive that the balance enquiry operation was removed completely from [136].

However, the retrenchment approach can handle the situation rather easily. The discrepancy in balances is just a reflection of what is a perfectly legitimate temporary disagreement in the balances during the playout of the protocol. The output relation of an A to B retrenchment of the balance enquiry operation can unproblematically relate the output discrepancy to the legitimate balance discrepancy (assuming that the protocol is indeed in its critical part). The retrenchment is validated by observing that since the protocol is entirely acceptable without the balance enquiry, adding the readonly balance enquiry operation, for which the output disagreements, such as they are, can be readily accounted for, cannot be objectionable. See [33]. Note that the balance enquiry scenario did not require the building of a tower as such. Rather, dealing with the balance discrepancy was more a question of retrenchment-enhancing an existing refinement in order to comfortably account for a state of affairs that was just a little too awkward to fall within the scope of the refinement technique used in [136].

10. Conclusions

In the preceding sections we surveyed the way the refinement has evolved to address not only the provision of guaranteed properties of specifications during the implementation of a contracted model, but also the desirability of acting as a specification constructor above it. In the latter arena, it sometimes proves restrictive regarding the relationships between models that it can express, and this provided the spur for the introduction of retrenchment, first introduced in [23] for the B-Method; (see also [119,118,120]). Retrenchment moves away from refinement, by abandoning the link between refinement's semantic foundations and its POs. Freed to consider a more accommodating

operation PO, retrenchment responds to the need for flexibility by incorporating a disjunctive option in the PO consequent (among other things). This turns out to be a rather drastic move in terms of the loss of desirable refinement properties — how nice it would be if propositional logic offered us a 'one tenth of a disjunction' connective, allowing a gentler departure from refinement. But there is no such thing.

Retrenchment has more parameters than refinement and (as we have formulated it here) defaults to pure state based refinement for suitable values of these parameters (e.g. $P = O = \mathsf{true}, C = \mathsf{false}$). Therefore it can be expected to enjoy a weaker theory than refinement, but to be more widely applicable. The discussions of stepwise simulation and of control theory bear out this view. Retrenchment thus permits the incorporation of models within a development path that could not be brought together using refinement. This gives developers more flexibility, and invites its use in much broader contexts of requirements engineering and model evolution than its original motivations suggested.

It has to be stressed that retrenchment must be deployed in an entirely glass box manner, where the incompatibilities between levels of abstraction that it permits are transparent and are properly validated by 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. By widening the applicability of formal techniques in this manner, it is anticipated that retrenchment can help to overcome the bad press that formal development methods have to some extent acquired, as expressed in the delightful epithet of [34]: 'formal methods are both oversold and under-used'.

Acknowledgements

Since the initial introduction of retrenchment in [23], many peoples' comments have helped the authors to both refine and retrench its presentation into what appears in this paper. The authors would 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 Soorensen, Bill Stoddart, Sam Valentine, Steve Vickers, and a number of anonymous referees.

References

[1] M. Abadi, L. Lamport, The existence of refinement mappings, Theoret. Comput. Sci. 82 (1991) 253–284.
[2] J.R. Abrial, The B-Book, Cambridge University Press, 1996.
[3] J.R. Abrial, Event based sequential program development: application to constructing a pointer program, in: D. Mandrioli, H. Araki, S. Gnesi (Eds.), Proc. FM-03, in: LNCS, vol. 2805, Springer, 2003, pp. 51–74.
[4] J.R. Abrial, B. Event, Cambridge University Press, 2007 (to appear).
[5] J.R. Abrial, D. Cansell, D. Méry, Refinement and reachability in event-B, in: H. Treharne, S. King, M. Henson, S. Schneider (Eds.), Proc. ZB-05, in: LNCS, vol. 3455, Springer, 2005, pp. 222–241.
[6] R. Alur, D. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (1994) 183–235.
[7] R. Alur, T. Henzinger, E. Sontag (Eds.), Hybrid Systems III: Verification and Control, in: LNCS, vol. 1066, Springer, 1996.
[8] R. Alur, G. Pappas (Eds.), Hybrid Systems: Computation and Control, in: LNCS, vol. 2993, Springer, 2004.
[9] B. Andrews, An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, Academic Press, 1986.
[10] R.J.R. Back, On correct refinement of programs, J. Comp. Sys. Sci. 23 (1981) 49–68.
[11] R.J.R. Back, A calculus of refinements for program derivations, Acta Inf. 25 (1988) 593–624.
[12] R.J.R. Back, R. Kurki-Suonio, Decentralisation of process nets with centralised control, in: Proc. 2nd ACM SIGACT-SIGOPS Symp. on Princ. Dist. Comp., ACM, 1983, pp. 131–142.
[13] R.J.R. Back, K. Sere, Superposition refinement of reactive systems, Form. Asp. Comp. 8 (1996) 324–346.
[14] R.J.R. Back, J. von Wright, Refinement calculus Part I: Sequential nondeterministic programs, in: W.-P. de Roever, G. Rozenberg (Eds.), Proc. REX Workshop, Stepwise Refinement of Distributed Systems, in: LNCS, vol. 430, Springer, 1989, pp. 42–66.
[15] R.J.R. Back, J. von Wright, Refinement Calculus, A Systematic Introduction, Springer, 1998.
[16] J.-C. Bajard, C. Frougny, P. Kornerup, J.-M. Muller (Eds.), Proc. Fourth Conference on Real Numbers and Computers. Schloss Dagstuhl, Germany, Dagstuhl Conference No. 00162, DMTCS, Springer, 2000.
[17] R. Banach, Maximally abstract retrenchments, in: Proc. IEEE ICFEM-00, 2000, pp. 133–142.
[18] R. Banach, Retrenchment and System Properties, 2003 (submitted for publication).
[19] R. Banach, M. Bozzano, Retrenchment, and the generation of fault trees for static, dynamic and cyclic systems, in: Proc. SAFECOMP-06, in: LNCS, vol. 4166, Springer, 2006, pp. 127–141.

[20] R. Banach, R. Cross, Safety requirements and fault trees using retrenchment, in: Proc. SAFECOMP-04, in: LNCS, vol. 3219, Springer, 2004, pp. 210–223.
[21] R. Banach, C. Jeske, Stronger Compositions for Retrenchments, and Feature Engineering (2002) (submitted for publication).
[22] R. Banach, C. Jeske, M. Poppleton, Composition Mechanisms for Retrenchments (2004) (submitted for publication).
[23] R. Banach, M. Poppleton, Retrenchment: An engineering variation on refinement, in: D. Bert (Ed.), Proc. B-98, in: LNCS, vol. 1393, Springer, 1998, pp. 129–147. See also: UMCS Tech. Rep. UMCS-99-3-2 http://www.cs.man.ac.uk/cstechrep.
[24] R. Banach, M. Poppleton, Retrenchment and punctured simulation, in: H. Araki, A. Gallway, K. Taguchi (Eds.), Proc. IFM-99, Springer, 1999, pp. 457–476.
[25] R. Banach, M. Poppleton, Sharp retrenchment, modulated refinement and punctured simulation, Form. Asp. Comp. 11 (1999) 498–540.
[26] R. Banach, M. Poppleton, Retrenchment, refinement and simulation, in: J. Bowen, S. Dunne, A. Galloway, S. King (Eds.), Proc. ZB00, in: LNCS, vol. 1878, Springer, 2000, pp. 304–323.
[27] R. Banach, M. Poppleton, Fragmented retrenchment, concurrency and fairness, in: Proc. IEEE ICFEM-00, 2000, pp. 143–151.
[28] R. Banach, M. Poppleton, Model based engineering of specifications by retrenching partial requirements, in: Proc. IEEE MBRE-01, 2001, pp. 1–8.
[29] R. Banach, M. Poppleton, Retrenching partial requirements into system definitions: A simple feature interaction case study, Req. Eng. J. 8 (2003) 266–288.
[30] R. Banach, M. Poppleton, C. Jeske, S. Stepney, Retrenching the purse: Finite sequence numbers, and the tower pattern, in: J. Fitzgerald, I. Hayes, A. Tarlecki (Eds.), Proc. FM-05, in: LNCS, vol. 3582, Springer, 2005, pp. 382–398.
[31] R. Banach, C. Jeske, M. Poppleton, S. Stepney, Retrenching the purse: finite exception logs, and validating the small, in: Proc. IEEE/NASA SEW30-06, 2006.
[32] R. Banach, C. Jeske, M. Poppleton, S. Stepney, Retrenching the purse: Hashing injective CLEAR codes, and security properties, in: Proc. IEEE ISOLA-06, 2006 (in press).
[33] R. Banach, M. Poppleton, C. Jeske, S. Stepney, Retrenching the purse: The Balance enquiry quandary, and generalised and (1,1) forward refinements, Fund. Inf. 77 (2007) 29–69.
[34] L.M. Barroca, J.A. McDermid, Formal methods: Use and relevance for the development of safety-critical systems, Comput. J. 35 (1992) 579–599.
[35] A. Blikle, The clean termination of iterative programs, Acta Inf. 16 (1981) 199–217.
[36] E. Börger, Why use evolving algebras for hardware and software engineering? in: M. Bartosek, J. Staudek, J. Wiedermann (Eds.), Proc. SOFSEM-95, in: LNCS, vol. 1012, Springer, 1995, pp. 236–271.
[37] E. Börger (Ed.), Specification and Validation Methods, Oxford University Press, 1995.
[38] E. Börger, High level system analysis and design using abstract state machines, in: D. Hutter, W. Stephan, P. Traverso, M. Ullmann (Eds.), Proc. FM-Trends-98, in: LNCS, vol. 1641, Springer, 1999, pp. 1–43.
[39] E. Börger, W. Schulte, A programmer friendly modular definition of the semantics of java, in: J. Alves-Foss (Ed.), Formal Syntax and Semantics of Java, in: LNCS, vol. 1523, Springer, 1998, pp. 353–404.
[40] E. Börger, W. Schulte, Modular design for the java virtual machine, in: Börger (Ed.), Architecture Design and Validation Methods, Springer, 2000, pp. 297–357.
[41] E. Börger, R. Stärk, Abstract State Machines, Springer, 2003.
[42] E. Boiten, J. Derrick, IO-Refinement in Z, in: Proc. Third BCS-FACS Northern Formal Methods Workshop, Ilkley, UK, B.C.S., 1998. http://www.ewic.org.uk/ewic/workshop/view.cfm/NFM-98.
[43] E. Boiten, J. Derrick, Grey box data refinement, in: J. Grundy, M. Schwenke, T. Vickers (Eds.), Proc. IRW&FMPacific-98, in: Disc. Maths. and Theor. Comp. Sci., Springer, 1998, pp. 45–59.
[44] E. Brinksma, G. Scollo, C. Steenbergen, LOTOS specifications, their implementations and their tests, in: B. Sarikaya, G. Bochmann (Eds.), Proc. IFIP Protocol Specification, Testing and Verification (PSTV6), Elsevier, North-Holland, 1987, pp. 349–360.
[45] E. Brinksma, A theory for the derivation of tests, in: S. Aggarwal, K. Sabnani (Eds.), Proc. IFIP Protocol Specification, Testing and Verification (PSTV-8), Elsevier, North-Holland, 1988, pp. 63–74.
[46] M. Broy, M. Wirsing, Partial abstract types, Acta Inf. 18 (1982) 47–64.
[47] M. Broy, M. Wirsing, Generalised heterogeneous algebras and partial interpretations, in: J. Gruska, B. Rovan, J. Wiedermann (Eds.), Proc. CAAP-83, in: LNCS, vol. 233, Springer, 1983, pp. 29–43.
[48] M. Broy, Partial interpretations of higher order algebraic types, in: Ausiello, Protasi (Eds.), Proc. MFCS-86, in: LNCS, vol. 159, Springer, 1986, pp. 1–34.
[49] M. Broy, K. Stølen, Specification and Development of Interactive Systems, Springer, 2001.
[50] J.A. Clark, S. Stepney, H. Chivers, Breaking the model: Finalisation and a taxonomy of security attacks, Technical Report YCS-2004-371, University of York, 2004.
[51] D. Coleman, J.W. Hughes, The clean termination of pascal programs, Acta Inf. 11 (1979) 195–210.
[52] D. Crocker, Safe object-oriented software: The verified design-by-contract paradigm, in: Redmill, Anderson (Eds.), Proc. Twelfth Safety-Critical Systems Symposium, Springer, 2004, pp. 19–41. See also http://www.eschertech.com/papers.
[53] J. Derrick, E. Boiten, Refinement in Z and Object-Z, Foundations and Advanced Applications, FACIT, Springer, 2001.
[54] J. Derrick, E. Boiten, Relational concurrent refinement, Form. Asp. Comp. 15 (2003) 182–214.
[55] J. Derrick, H. Bowman, E. Boiten, M. Steen, Comparing LOTOS and Z refinement relations, in: Proc. FORTE/PSTV-9, Chapman and Hall, 1996, pp. 501–516.
[56] W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, J. Zwiers, Concurrency Verification, Cambridge University Press, 2001.

[57] W.-P. de Roever, K. Engelhardt, Data Refinement: Model-Oriented Proof Methods and their Comparison, Cambridge University Press, 1998.
[58] M.D. Di Benedetto, A. Sangiovanni-Vincentelli (Eds.), Hybrid Systems: Computation and Control, in: LNCS, vol. 2034, Springer, 2001.
[59] E.W. Dijkstra, Notes on structured programming, in: Structured Programming, Academic Press, 1972.
[60] E.W. Dijkstra, C.S. Scholten, Predicate Calculus and Program Semantics, Springer, 1990.
[61] H. Doornbos, A relational model of programs without the restriction to egli-milner constructs, in: E.-R. Olderog (Ed.), Proc. PROCOMET-94, IFIP, 1994, pp. 357–376.
[62] D.T.I. Information Technology Security Evaluation Criteria Department of Trade and Industry. 1991. http://www.cesg.gov.uk/site/iacs/itsec/media/formal-docs/Itsec.pdf.
[63] S. Dunne, A case for general correctness. Sch. Comp. Math. Univ. Teeside Preprint, 2001.
[64] S. Dunne, Recasting Hoare and He's Relational theory of programs in the context of general correctness, in: A. Butterfield, G. Strong, C. Pahl (Eds.), Proceedings of the 5th Irish Workshop in Formal Methods, IWFM 2001, Workshops in Computing, British Computer Society, 2001. http://ewic.bcs.org/conferences/2001/5thformal/papers.
[65] S. Dunne, A. Galloway, B. Stoddart, Specification and Refinement in General Correctness, in: Proc. Third BCS-FACS Northern Formal Methods Workshop, Ilkley, U K, B.C.S., 1998 http://www.ewic.org.uk/ewic/workshop/view.cfm/NFM-98.
[66] H. Ehrig, M. Grosse-Rhode, Functorial theory of parameterised specifications in generalised specification frameworks, Theoret. Comput. Sci. 135 (1994) 221–266.
[67] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification I, Springer, 1985.
[68] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification II, Springer, 1990.
[69] J. Fiadeiro, T. Maibaum, Categorical semantics of parallel program, Design. Sci. Comp. Prog. 28 (1997) 111–138.
[70] J. Fitzgerald, P.G. Larsen, Modelling Systems: Practical Tools and Techniques in Software Development, Cambridge University Press, 1998.
[71] J. Fleuriot, A Combination of Geometry Theorem Proving and Nonstandard Analysis, with Application to Newton's Principia, Springer, 2001. Also Ph.D. Thesis, Cambridge University Computing Laboratory 1999, see also Cambridge University Computing Laboratory Technical Report No. 442.
[72] N. Francez, I.R. Forman, Superimposition for interactive processes, in: J. Baeten, J.-W. Klop (Eds.), Proc. CONCUR-90, in: LNCS, vol. 458, Springer, 1990, pp. 230–245.
[73] M.L. Franklin, G.F. Powell, J.D. Workman, Digital Control of Dynamic Systems, Addison Wesley Longman, 1998.
[74] J.A. Goguen, J. Meseguer, Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations, Theoret. Comput. Sci. 105 (1992) 217–273.
[75] J.R. Harrison, Theorem Proving with the Real Numbers, Springer, 1998, Also Ph.D. Thesis, Cambridge University Computing Laboratory 1996, also Cambridge University Computing Laboratory Technical Report No. 408.
[76] I.J. Hayes, Specification Case Studies, 2nd ed., Prentice-Hall, 1993.
[77] I.J. Hayes, J.W. Sanders, Specification by interface separation, Form. Asp. Comp. 7 (1995) 430–439.
[78] E.C.R. Hehner, A Practical Theory of Programming, Springer, 1993.
[79] T. Hentzinger, S. Sastry, Hybrid Systems: Computation and Control, in: LNCS, vol. 1386, Springer, 1998.
[80] F.B. Hildebrand, Advanced Calculus for Applications, Prentice-Hall, 1962.
[81] C.A.R. Hoare, Proof of correctness of Data representations, Acta Inf. 1 (1972) 271–281.
[82] C.A.R. Hoare, H. Jifeng, Unifying Theories of Programming, Prentice-Hall, 1998.
[83] W. Hodges, Model Theory, Cambridge University Press, 1993.
[84] International Standards Organisation, Common Criteria for Information Security Evaluation, ISO 15408, v. 3.0 rev. 2, 2005.
[85] H. Jeffreys, B.S. Jeffreys, Methods of Mathematical Physics, 3rd ed., Cambridge University Press, 1956.
[86] C. Jeske, Algebraic integration of retrenchment and refinement, Ph.D. Thesis, Manchester University, Department of Computer Science, 2005.
[87] C. Jeske, R. Banach, Minimally and maximally abstract retrenchments, in: M. Butler, L. Petre, K. Sere (Eds.), Proc. IFM-02, in: LNCS, vol. 2335, Springer, 2002, pp. 380–399.
[88] C.B. Jones, Tentative steps towards a development method for interfering programs, ACM Trans. Prog. Lang. Sys. 5 (1983) 596–619.
[89] C.B. Jones, Systematic Software Development Using VDM, 2nd ed., Prentice-Hall, 1990.
[90] C.B. Jones, R.C. Shaw, Case Studies in Systematic Software Development, Prentice-Hall, 1990.
[91] S. Kamin, M. Archer, Partial implementations of abstract data types: A dissenting view, in: G. Kahn, D. MacQueen, G. Plotkin (Eds.), Proc. Semantics of Data Types, in: LNCS, vol. 173, Springer, 1984, pp. 317–336.
[92] S. Katz, A superimposition control construct for distributed systems, ACM Trans. Prog. Lang. Sys. 15 (1993) 337–356.
[93] B. Krogh, N. Lynch (Eds.), Hybrid Systems: Computation and Control, in: LNCS, vol. 1790, Springer, 2000.
[94] B.C. Kuo, Digital Control Systems, 2nd ed., Oxford University Press, 1992.
[95] L. Lamport, A simple approach to specifying concurrent systems, Comm. ACM 32 (1989) 32–45.
[96] L. Lamport, A temporal logic of actions, ACM Trans. Prog. Lang. Sys. 16 (1994) 872–923.
[97] K. Lano, H. Haughton, Specification in B: An Introduction Using the B-Toolkit, Imperial College Press, 1996.
[98] D. Lester, Using PVS to validate the inverse trigonometric functions of an exact arithmetic, in: R. Alt, A. Frommer, R. Kearfott, W. Luther (Eds.), Numerical Software with Result Verification, in: LNCS, vol. 2991, Springer, 2003, pp. 259–273.
[99] B. Liskov, J.M. Wing, A behavioural notion of subtyping, ACM Trans. Prog. Lang. Sys. 16 (1994) 1811–1841.
[100] S. Liu, Evolution: A More practical approach than refinement for software development, in: Proc. ICECCS-97, IEEE, 1997, pp. 142–151.
[101] B. Mahony, The specification and refinement of timed processes, Ph.D. Thesis, Dept. of Computer Science, Queensland University, 1992.
[102] O. Maler (Ed.), Hybrid and Real-Time Systems, in: LNCS, vol. 1201, Springer, 1997.
[103] O. Maler, A. Pnueli (Eds.), Hybrid Systems: Computation and Control, in: LNCS, vol. 2623, Springer, 2003.
[104] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer, 1992.

[105] B. Meyer, Object Oriented Construction, Prentice-Hall, 1988.
[106] R. Miarka, E. Boiten, J. Derrick, Guards, preconditions and refinement in Z, in: J. Bowen, S. Dunne, A. Galloway, S. King (Eds.), Proc. ZB-00, in: LNCS, vol. 1878, Springer, 2000, pp. 286–303.
[107] A. Mikhajlova, E. Sekerinski, Class refinement and interface refinement in object-oriented programs, in: J. Fitzgerald, C. Jones, P. Lucas (Eds.), Proc. FME-97, in: LNCS, vol. 1313, Springer, 1997, pp. 82–101.
[108] C. Morgan, Programming from Specifications, 2nd ed., Prentice-Hall, 1994.
[109] J.M. Morris, A theoretical basis for stepwise refinement and the programming calculus, Sci. Comp. Prog. 9 (1987) 287–306.
[110] J.M. Morris, A. Bunkenburg, Partiality and nondeterminacy in program proofs, Form. Asp. Comp. 10 (1998) 76–96.
[111] J.M. Morris, A. Bunkenburg, Term transformer semantics, ACM Trans. Prog. Lang. Sys. (2000) (submitted for publication).
[112] C. Munoz, D. Lester, Real number calculations and theorem proving, in: J. Hurd, T. Melham (Eds.), TPHOLs05, in: LNCS, vol. 3603, Springer, 2005, pp. 195–210.
[113] M. Nielsen, K. Havelund, K.R. Wagner, C. George, The RAISE language method and tools, Form. Asp. Comp. 1 (1989) 85–114.
[114] D.S. Neilson, From Z to C: Illustration of a rigorous development Method, Ph.D. Thesis, Oxford University Computing Laboratory Programming Research Group, Technical Monograph PRG-101, 1990.
[115] G. Nelson, A generalisation of Dijkstra's calculus, ACM Trans. Prog. Lang. Sys. 11 (1989) 517–561.
[116] O. Owe, An Approach to Program Reasoning Based on a First Order Logic for Partial Functions, University of Oslo Institute of Informatics Research Report No. 89, ISBN: 82-90230-88-5, 1985.
[117] O. Owe, Partial logics reconsidered: a conservative approach, Form. Asp. Comp. 3 (1993) 1–16.
[118] M.R. Poppleton, Formal Methods for Continuous Systems: Liberalising Refinement in B, Ph.D. Thesis, Manchester University, Department of Computer Science, 2001.
[119] M.R. Poppleton, R. Banach, Retrenchment: Extending the reach of refinement, in: Proc. IEEE ASE-99, 1999, pp. 158–165.
[120] M.R. Poppleton, R. Banach, Retrenchment: Extending Refinement for Continuous and Control Systems, in: Proc. IWFM-00, 2000, 26pp., http://ewic.org.uk/ewic/workshop/view.cfm/IWFM-00.
[121] M.R. Poppleton, R. Banach, Controlling control systems: An application of evolving retrenchment, in: D. Bert (Ed.), Proc. BZ-02, in: LNCS, vol. 2272, Springer, 2002, pp. 42–61.
[122] Retrenchment Homepage; http://www.cs.man.ac.uk/retrenchment.
[123] RODIN. The RODIN Project. Rigorous Open Development Environment for Complex Systems, 2004 http://rodin.cs.ncl.ac.uk/index.html.
[124] G. Sambin, in: D. Skordev (Ed.), Intuitionistic Formal Spaces – a First Communication. Mathematical Logic and its Applications, Plenum, New York, 1987, pp. 187–204.
[125] G. Sambin, S. Gebellato, A preview of the basic picture: A new perspective on formal topology, in: T. Altenkirch, W. Naraschewski, B. Reus (Eds.), Proc. TYPES-98, in: LNCS, vol. 1657, Springer, 1999, pp. 194–207.
[126] G. Schellhorn, Verification of abstract state machines, Ph.D. Thesis, University of Ulm Fakultät für Informatik, 1999.
[127] G. Schellhorn, Verification of ASM refinements using generalized forward simulation, JUCS 7 (2001) 952–979.
[128] F.B. Schneider, On Concurrent Programming, Springer, 1997.
[129] E. Sekerinski, K. Sere, Program Development by Refinement: Case Studies Using the B Method, Springer, 1998.
[130] G. Smith, Stepwise development from ideal specifications, in: Proc. IEEE ACSC-00, Aus. Comp. Sci. Comm. 22 (2000) 227–233.
[131] G. Smith, C. Fidge, Incremental development of real-time requirements: The light control case study, JUCS 6 (2000) 704–730.
[132] J.M. Spivey, The Z Notation: A Reference Manual, 2nd ed., Prentice-Hall, 1993.
[133] Y.V. Srinivas, R. Jullig, Specware(TM): Formal support for composing software, in: B. Moller (Ed.), Proc. MPC-95, in: LNCS, vol. 947, Springer, 1995, pp. 399–422. Also: Kestrel Institute Technical Report KES.U.94.5 http://www.kestrel.edu.
[134] R. Stärk, J. Schmid, E. Börger, Java and the Java Virtual Machine, Definition, Verification, Validation, Springer, 2001.
[135] S. Stepney, D. Cooper, J. Woodcock, More powerful Z data refinement: Pushing the state of the art in industrial refinement, in: J. Bowen, A. Fett, M. Hinchey (Eds.), Proc. ZUM-98, in: LNCS, vol. 1493, Springer, 1998, pp. 284–307.
[136] S. Stepney, D. Cooper, J. Woodcock, An electronic purse: specification, refinement and proof, Technical Report PRG-126, Oxford University Computing Laboratory, 2000.
[137] B. Strulo, How firing conditions help inheritance, in: J. Bowen, M. Hinchey (Eds.), Proc. ZUM-95, in: LNCS, vol. 967, Springer, 1996, pp. 264–275.
[138] W. Swartout, R. Balzer, On the inevitable intertwining of specification and implementation, Comm. ACM 25 (1982) 438–440.
[139] F. Vaandrager, J. van Schuppen, Hybrid Systems: Computation and Control, in: LNCS, vol. 1569, Springer, 1999.
[140] D. van Dalen, Logic and Structure, 3rd ed., Springer, 1997.
[141] P.H.J. van Eijk, C.A. Vissers, M. Diaz, The Formal Specification Technique LOTOS, Elsevier, North Holland, 1989.
[142] R. Waldinger, L. Blaine, D. Spinoza, L.-M. Gilham, A. Goldberg, C. Green, R. Jullig, J. Liu, J. McDonald, D.R. Smith, Y.V. Srinavas, T.C. Wang, S. Westfold, Specware Language Manual, Kestrel Institute, 1998, http://www.kestrel.edu.
[143] H. Wehrheim, Behavioral subtyping relations for active objects, Form. Methods Syst. Des. 23 (2003) 143–170.
[144] K. Weihrauch, An Introduction to Computable Analysis, Springer, 2000.
[145] N. Wirth, The development of programs by stepwise refinement, Comm. ACM 14 (1971) 221–227.
[146] J. Woodcock, J. Davies, Using Z: Specification, Refinement, and Proof, Prentice-Hall, 1996.
[147] J.B. Wordsworth, Software Engineering with B, Addison-Wesley, 1996.
[148] J. von Wright, The lattice of data refinement, Acta Inf. 31 (1994) 105–135.